НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

«До захисту допущено»
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6. 050201 «Системна інженерія»

на тему: «Моделювання автоматизованої системи керування на базі технології XML»

Виконав (-ла):
студент (-ка) IV курсу, групи ІА-52
Калюх Олег Миколайович _____

Керівник:
старший викладач Яланецький В.А.
_____

Рецензент:
_____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2019 рік

АНОТАЦІЯ

Калюх О.М. Моделювання автоматизованої системи керування на базі технології XML. КПІ ім. Ігоря Сікорського, Київ, 2019.

Проект містить 86 сторінок, 85 рисунків, 1 додаток, посилання на 16 літературних джерел та 4 конструкторські документи.

Ключові слова: автоматизована система керування, моделювання, XML, OpenModelica, Modelica, автоматизація.

Об’єктом розробки є модель автоматизованої системи керування.

Мета розробки – дослідження засобів моделювання автоматизованих систем керування, з подальшим експортом моделей в XML.

У дипломному проекті розроблено моделі в автоматизованих системах керування, а саме: модель системи з електричним, механічним, теплотехнічним та гідродинамічним процесами. Побудовані моделі експортовано в XML.

Одержані результати можуть бути корисними при моделюванні автоматизованих систем керування.

SUMMARY

Kaliukh O.M. Design of automated control system based on XML technology. Igor Sikorsky KPI, Kyiv, 2019.

The project contains 86 pages, 85 images, 1 attachment, links to 16 literary sources and 4 design documents.

Keywords: automated control system, modeling, XML, OpenModelica, Modelica, automation.

The object of development is the model of automated control system.

The purpose of the development – study the means of modeling automated control systems, followed by the export of models in XML.

The graduation project developed models in automated control systems, namely: a model of a system with electric, mechanical, heat engineering and hydrodynamic processes. The constructed models are exported to XML.

The results obtained can be useful in the simulation of automated control systems.

Пояснювальна записка
до дипломного проекту
на тему: «Моделювання автоматизованої системи керування на базі технології XML»

Київ – 2019 рік

Зміст

ВСТУП

Автоматизована система керування - це комплекс апаратних і програмних дій, а також, постійного складу робочого персоналу, який має на меті управління різними процесами в рамках технологічного процесу. [1] В наш час жодна АСК не будується та не впроваджується без перевіреної та повністю математично побудованої моделі. Найважливішею задачею АСК є підвищення якості управління об'єкта на основі росту продуктивності праці і удосконалення методів планування процесів керування. А також зменшення кількості рішень, які приймає людина.

З точки зору моделювання АСК та моделювання загалом, XML можна розглядати як кінцеву форму-представлення будь-якої моделі. Транслювавши модель в XML ми отримаємо готовий код, який можна використовувати в різних середовищах моделювання. Наприклад модель об’єкта управління можна побудувати на базі мови modelica, а діаграму вимог - засобами SysML.

Мета дипломного проекту – знайти безкоштовний, загальнодоступний програмний засіб для моделювання АСК. Даний програмний засіб повинен давати можливість будувати моделі об’єктів керування, керуючих пристроїв. Також цей засіб повинен мати можливість представляти готові моделі в форматі XML.

Постановка завдання:

- дослідити поняття АСК;
- розглянути мову розмітки XML, як сучасний засіб для моделювання, в тому числі для моделювання АСК;
- дослідити існуючі програмні продукти для моделювання, з подальшим експортом моделі в XML;
- знайти загальнодоступний програмний засіб для моделювання АСК, об’єктів керування та керуючих пристроїв;
- побудувати імітаційну модель електричної схеми;

- побудувати імітаційну модель механічної системи;
- побудувати імітаційну модель теплотехнічного процесу в АСК;
- побудувати імітаційну модель гідродинамічної АСК;
- експортувати побудовані моделі в формат XML.

1 ПОНЯТТЯ ТА ВИМОГИ ДО АСК

Автоматизована система керування - це комплекс апаратних і програмних дій, а також, постійного складу робочого персоналу, який має на меті управління різними процесами в рамках технологічного процесу, виробництва [1]: збір інформації з первинних або передавальних перетворювачів сигналів для розрахунку, видачі та реалізації керуючих впливів на об'єкт керування(ОК) відповідно до критеріїв керування системи. АСК використовують у різних галузях виробництва: промисловості, енергетикі, транспорті і тд. Автоматизована система керування відрізняється від автоматичної системи керування тим, що в першій зберігаються за людиною-оператором деякі функції, які не піддаються автоматизації або не є доцільними.

Найважливішою задачею АСК є підвищення якості управління об'єкта на основі росту продуктивності праці і удосконалення методів планування процесів керування. А також зменшення кількості рішень, які приймає людина.

Функції АСК встановлюють в технічному завданні на створення конкретної автоматичної системи управління на основі аналізу цілей управління, заданих ресурсів для їх досягнення, очікуваного ефекту від автоматизації та відповідно до стандартів, що поширюються на даний вид АСК. Кожна функція АСК реалізується сукупністю комплексів завдань, окремих завдань і операцій. Функції в загальному випадку включають в себе наступні елементи (дії): планування і прогнозування, облік, контроль, аналіз, координацію і регулювання. [1]

1.1 Моделювання АСК

Моделювання є потужним засобом аналізу і синтезу складних об'єктів, процесів і явищ. Рішення багатьох складних наукових і технічних завдань значно спрощується при моделюванні, тобто при спрощенні і заміні одних об'єктів іншими, що забезпечують відображення найбільш істотних для дослідника властивостей і особливостей спрощення об'єктів. В даний час,

моделювання знаходить широке застосування в автоматиці, обчислювальній та вимірювальній техніці, радіотехніці і связуванні математики та інших областей науки і техніки. [2]

Моделювання полягає у виявленні основних властивостей досліджуваного процесу, побудові моделей і їх застосування для прогнозування поведінки системи. Критерієм правильності моделювання є практика.

Цінність методів моделювання полягає в тому, що вони дозволяють істотно скоротити і полегшити людську роботу, який зазвичай дорога і складна, а також збільшити достовірність математичного опису і розрахунків. Моделлю можна відтворити реальний технічний пристрій і абстрактний математичний опис, що відображає реальний стан речей. Застосування технічних засобів надає моделюванню експериментальний характер, а модель математичного опису теоретично розкриває характер явища.

В даний час для динамічних досліджень систем автоматичного керування(АСК) широко використовуються методи машинного моделювання на основі електронних обчислювальних машин: цифрових(ЦОМ), аналогових(АОМ) та аналого-цифрових комплексів(АЦОК).

З моделюванням АСК тісно пов'язана проблема ідентифікації. Під ідентифікацією в широкому сенсі розуміється отримання або уточнення за експериментальними даними моделі реального об'єкта, вираженою в тих чи інших термінах. Як видно, ідентифікація в широкому сенсі становить невід'ємну частину всякої справжньої науки і має давнє походження.

В даний час у зв'язку з пред'явленням все більш високих вимог до процесів управління в різних областях техніки проблема ідентифікації стає винятково важливою.

Не можна забезпечити якісне управління системою, якщо невідома з достатньою точністю її математична модель. Для побудови математичних моделі можуть бути використані як теоретичні, так і експериментальні методи. Досвід, накопичений при проектуванні систем управління, переконливо

свідчить про те, що не можна побудувати матетичну модель, співставну реальній системі, тільки на основі теоретичних досліджень фізичних процесів в системі. Тому в процесі проектування систем управління одночасно з теоретичними дослідами проводяться численні досліди по визначенню і уточненню математичної моделі системи, тобто її ідентифікація.

При вирішенні задачі ідентифікації вважається, що вся структура системи і класи моделей, до яких вони належать досить обширна. Така постановка задачі ідентифікації найбільш відповідає реальним умовам проектування і широко використовується в інженерній практиці.

1.2 Мова розмітки XML

XML - це мова розмітки, що призначена для передачі даних, а не для їх відображення. Основною задачею є відображення даних, акцентуючи увагу на тому, як ці дані відображаються. Тобто, XML - це мова, яка допомагає транспортувати інформацію. [3]

Розробка мови робилась таким чином, щоб мати простий формальний синтаксис, який буде зручним для створення і обробці документів программ і одночасно буде зручно для читання документу людиною. XML вважається розширеною мовою через те, що він не фіксує розмітку, яка була використана в документах.

З програмної сторони документи складається із сутностей(найменша одиниця документа), яка може перенаправляти на іншу сутність. Єдиний кореневий елемент - це документальна сутність, яка є обов'язковою частиною документа, яка може включати в собі вкладені в неї документи і символічні дані. Зміст сутності - символічні дані.

З логічної точки зору документ складається з коментарів, об'яв, елементів, направлень на сутність і інструкцій по обробці. Все це в документі структуризується розміткою.

Для корректного відображення XML документа необхідно, щоб виконувались наступні вимоги:

- правильне оформлення документу
- дотримання всіх правил розмітки

З точки зору моделювання АСК та моделювання загалом, XML можна розглядати як кінцеву форму-представлення будь-якої моделі. Транслювавши модель в XML ми отримаємо готовий код, який можна використовувати в різних середовищах моделювання. Наприклад модель об’єкта управління можна побудувати на базі мови modelica, а діаграму вимог засобами SysML.

Висновки до розділу 1

В даному розділі були розглянуті основні поняття та вимоги автоматичної системи керування та використання АСК в різних галузях виробництва. В результаті було виявлено, що з моделюванням АСК пов'язана проблема ідентифікації, отримання за експериментальними даними моделі реального об'єкта. Тобто, не можливе якісне керування системою, якщо невідома з достатньою точністю математична модель системи. Розглянули мову розмітки XML, як сучасний засіб для моделювання, в тому числі для моделювання АСК.

2 ОГЛЯД ІСНУЮЧИХ ЗАСОБІВ ДЛЯ МОДЕЛЮВАННЯ АСК ТА ЇХ ПОДАЛЬШОГО ЕКСПОРТУ В XML

2.1 Засоби на базі мови моделювання SysML

SysML - це предметно-орієнтована мова моделювання систем. Підтримує визначення, аналіз, проектування, перевірки та підтвердження відповідності широкого спектра систем. SysML спочатку розроблявся в рамках проекту специфікації з відкритим вихідним кодом, і має відкриту ліцензію для поширення і використання. Як мова, SysML є розширенням частини мови UML. [4]

Орієнтованим на моделювання програмних продуктів, SysML надає системному інженерові додаткові можливості яких не може представити UML: Діаграми вимог - для збору вимог та параметрична діаграма - для кількісного аналізу та/або аналізу продуктивності.

Загалом діаграми SysML описують чотири основні області:

- Структурна діаграма (Structure) - архітектурні елементи (логічні та фізичні), "Блоки", а також їх взаємозв'язки: діаграми визначення блоків, діаграми внутрішніх даних, діаграми пакетів.

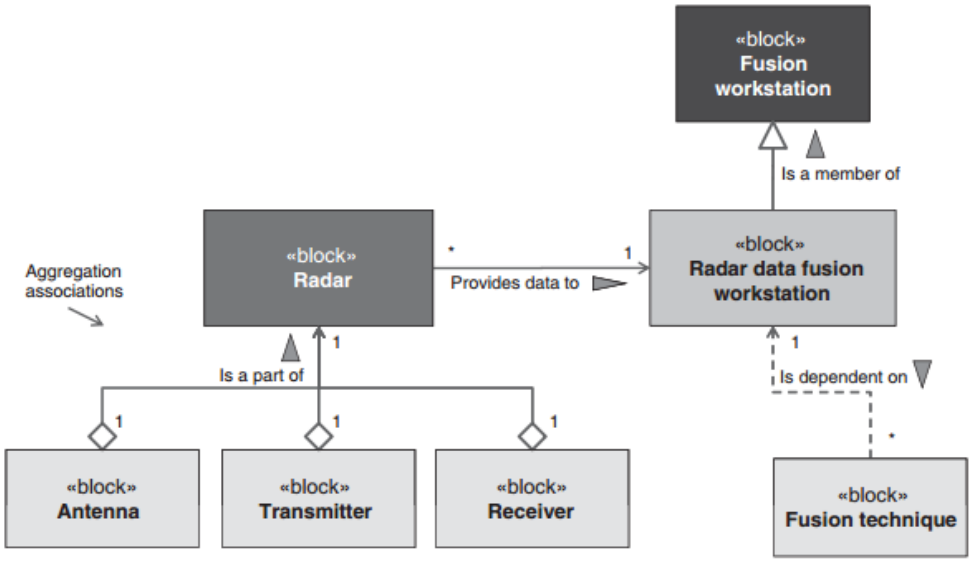

Рисунок 2.1 – Приклад структурної діаграми в SysML [5]

- Діаграма поведінки (Behavioral) - те, як поводиться система, включаючи зміну станів, послідовності активностей, функції та взаємодії. Додані чотири розширення: потік управління доповнений керуючим оператором; за допомогою безперервних потоків об'єктів стало можливо моделювати безперервні системи; з потоками можна асоціювати ймовірності; розширені правила моделювання діяльності.

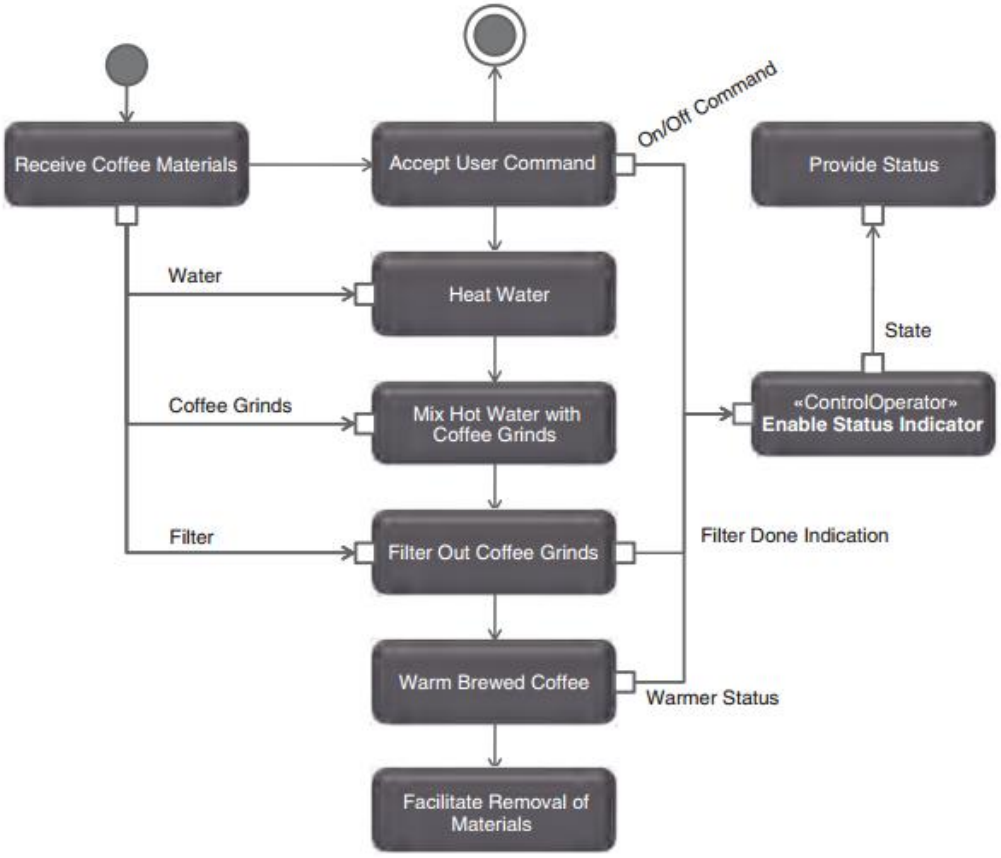

Рисунок 2.2 – Приклад діаграми поведінки в SysML [5]

- Діаграма вимог (Requirements) - показує системні вимоги та їх зв'язки з іншими елементами. Корисно для розробки вимог, включаючи валідацію та верифікацію (V&V).

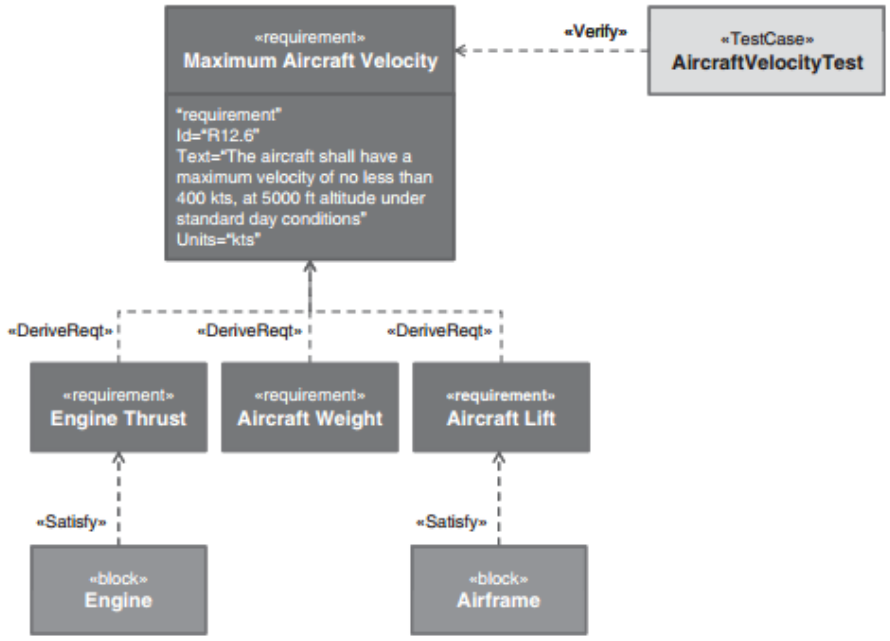

Рисунок 2.3 – Приклад діаграми вимог в SysML [5]

- Параметрична діаграма (Parametrics) - показує параметричні обмеження між структурними елементами. Корисний для аналізу продуктивності та кількісного аналізу.

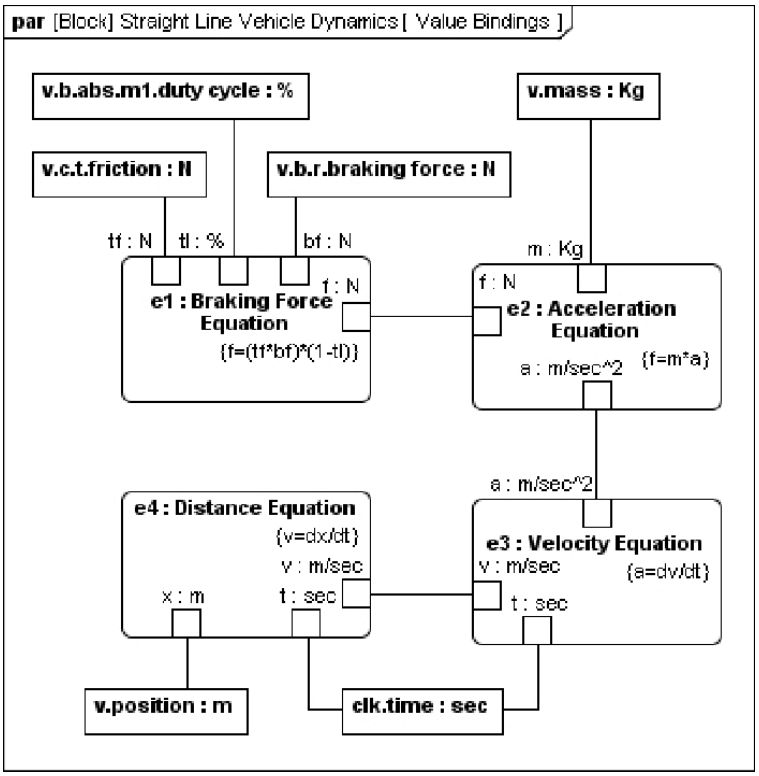

Рисунок 2.4 – Приклад параметричної діаграми в SysML [6]

Визначено види таких відповідностей (dependency relationships) в SysML:

- verify (наприклад, до тестового сценарієм або іншого елементу, що може визначити відповідність),
- satisfy (до елементу, який вже визначає відповідність),
- refine
- trace
- copy і т.д.

У цих відповідностей може бути вказано обгрунтування (rationale). Вказується також трасування (залежність від вимоги) і уточнення, а також производность вимог один від одного.

Вимоги в SysML можуть бути зібрані в таблиці - головним чином для зручності сприйняття.

Важлива ідея повторного використання вимог: тому вводиться відношення копіювання - вимоги-оригінали (master, тобто "оригінали") і вимоги-копії (slave, тобто "копії"), які суть випадки повторного використання вимог-оригіналів.

Головне використання вимог в SysML - це анотація вимогами архітектури, вираженої в SysML. Звичайно, при цьому є досвід меппинга вимог SysML в RIF, бо мало кого влаштовує мати вимоги прив'язаними саме до SysML-поданням всіх інших архітектурних, дизайнерських, тестіровочного артефактів. Втім, OMG веде активні узгодження RIF (стара назва ReqIF), SysML і AP 233 - і при цьому доводиться доповнювати SysML, щоб задовольнити RIF. Все це одне і те ж за великим рахунком: анотація архітектури обривками тексту і контроль конфігурації цих уривків.

Прив'язка (allocation) - механізм SysML, що дозволяє поєднувати елементи різних моделей. У саму мову вмонтовані три типи прив'язок:

- прив'язка поведінки - пов'язує поведінку (представлене на одній або декількох поведінкових діаграмах) з блоком, який це поведінка реалізує
- прив'язка структури - з'єднує логічні структури з фізичними

- прив'язка потоку об'єктів - пов'язує потік елементів (на структурній діаграмі) з дугою потоку об'єкта (на діаграмі діяльності).

2.2 Відкритий стандарт AutomationML

AutomationML є нейтральним форматом даних, що базується на XML для зберігання та обміну інженерною інформацією заводу, яка надається як відкритий стандарт. Метою AutomationML є взаємозв'язок гетерогенного інструментального ландшафту сучасних інженерних засобів у різних дисципліна. Наприклад, машинобудівний завод, електротехнічне проектування, розробка HMI, PLC, робот-контроль. [7]

AutomationML описує реальні компоненти установки як об'єкти, що інкапсулюють різні аспекти. Об'єкт може складатися з інших суб-об'єктів і сам може бути частиною більшої композиції. Він може описувати гвинт, пазуру, робота або повну виробничу комірку різного рівня деталізації. Типові об'єкти в автоматизації заводів містять інформацію про топологію, геометрію, кінематику і логіку, де логіка включає в себе послідовність, поведінку і контроль.

AutomationML включає різні стандарти через сильно типізовані посилання у форматах:

- Топологія, реалізована з CAEX (IEC 62424). Властивості та відносини об'єктів у їх ієрархічній структурі
- Геометрія реалізована з COLLADA групи «Хронос». Графічні атрибути та 3D-інформація
- Кінематика реалізована з COLLADA. Підключення та залежності між об'єктами для підтримки планування руху
- Логіка реалізована за допомогою PLCopen XML. Послідовності дій, внутрішня поведінка об'єктів та сполуки вводу / виводу [8]

Для майбутніх розширень AutomationML розроблено для інтеграції інших форматів, використовуючи той самий механізм посилання.

2.3 Засоби на базі мови Modelica

Modelica є об'єктно-орієнтованою мовою моделювання для опису звичайних і диференційних алгебраїчних рівнянь, системи поєднуються з дискретними подіями, так званий гібрид DAEs. Такі моделі ідеально підходять для представлення фізичної поведінки і обміну енергії, сигналів або інших безперервних взаємодій між компонентами системами. [9]

Моделі Modelica подібні за структурою до SysML-моделей в тому сенсі, що моделі Modelica складаються з композицій підмоделей, з'єднаних портами, які представляють потік сигналу. Моделі є актуальними, заснованими на рівняннях і декларативними. Мова Modelica визначається і підтримується Асоціацією Modelica, яка публікує офіційну специфікацію.

Моделі в Modelica охоплюють області, починаючи від (аналогових і цифрових) електричних систем, механічних і теплових систем, блок-схем для управління. Нарешті, також варто зазначити, що в рамках спільноти Modelica існує багато напрямків з розробки відкритих (“open source”) рішень, таких як у проекті OpenModelica.

2.3.1 Мова графічного моделювання ModelicaML

ModelicaML - це графічна мова моделювання. ModelicaML реалізовано як профіль UML. [10]

Даний профіль визначає розширення мета-моделі UML, яка необхідна для захоплення конструкцій Modelica. Профілі UML (Stereotypes) дозволяють розширювати моделі UML, щоб представити поняття, які відсутні в UML або є вузькоспеціалізованими. Крім того, з точки зору UML, Modelica є специфічною мовою. Для інтеграції специфічних мов, UML визначає конструкції, які чудово використовуються в ModelicaML. Термін «мета-модель» використовується в світі графічного моделювання як аналог терміну «граматика» в інших мовах програмування. Як і будь-яка граматика, мета-модель визначає мовні поняття

(класи) і взаємозв'язки між ними. ModelicaML призначений для генерації виконуваного файлу коду Modelica з графічно побудованих моделей. [11]

Метою ModelicaML є надання можливості моделювати складні виконувані конструкції. Однак, як UML, так і SysML і ModelicaML є лише графічним зображенням коду. Моделі ModelicaML в кінцевому рахунку транслюються в код Modelica. Отже, семантика виконання визначається мовою Modelica і, зрештою, компілятором Modelica, що буде перекладати згенерований код Modelica у виконувану форму.

Існує різниця між графічним поданням, наданим поточними інструментами Modelica і нотацію ModelicaML. Інструменти Modelica надають одну діаграму. Діаграма з'єднання показує клас Modelica компоненти (як правило, зображені як конкретні іконки з роз'ємами) класу і їх взаємоз'єднання (підключення). Графічне позначення визначається людиною, яка будує і не стандартизоване специфікацією мови; це зазвичай специфічні для області застосування.

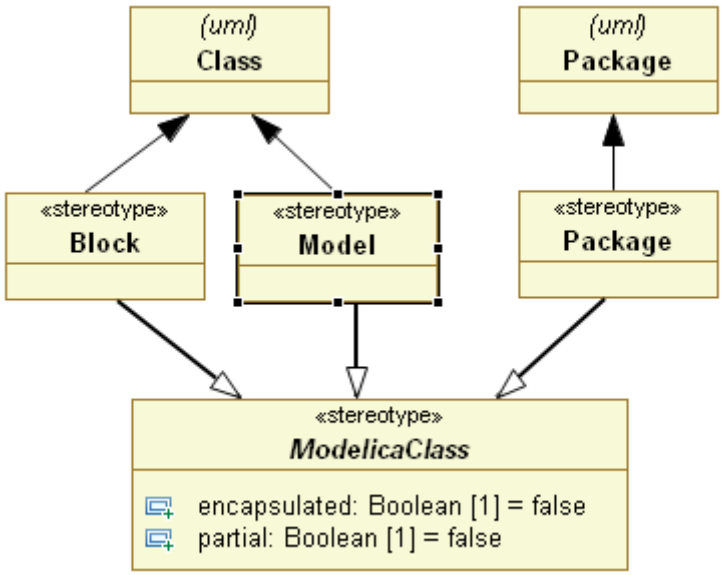

Рис 2.5. Діаграма з'єднання [12]

Профіль ModelicaML для Eclipse призначений для використання людьми:

- які знайомі з UML/SysML і хотіли б скористатися перевагами семантики ModelicaML для побудови моделей
- які звикли до Modelica і хотіли б скористатися перевагами графічного моделювання замість традиційного текстового підходу для створення моделей систем.

2.3.2 Платформа з відкртим кодом JModelica.org

JModelica.org - безкоштовна і відкрита програмна платформа, заснована на мові Modelica для моделювання, оптимізації та аналізу складних динамічних систем. Платформа підтримується і розвивається Modelon AB у співпраці з академічними та промисловими установами, зокрема з Університетом Лунд та Центром контролю складних систем у Лунді. Платформа була використана в промислових проектах із застосуванням в робототехніці, системах транспортних засобів, енергетичних системах. [13]

Ключовими компонентами платформи є:

- Компілятор Modelica для перекладу вихідного коду Modelica у код C або XML. Компілятор також генерує моделі, які відповідають стандарту функціонального макету інтерфейсу.
- Пакет Python для моделювання динамічних моделей, Assimulo. Assimulo - пакет моделювання для розв'язання звичайних рівнянь і використовується як двигун моделювання в JModelica.org.
- Алгоритми розв'язування великомасштабних динамічних задач оптимізації з використанням локальних методів колокації на скінченних елементах та методах псевдо спектральної колокації.
- Пакет Python для взаємодії з користувачем. Усі частини платформи доступні з Python, включаючи компіляцію та завантаження моделей, моделювання та оптимізацію.
- Плагін Eclipse для редагування вихідного коду Modelica. [14]

JModelica.org підтримує мову моделювання Modelica для моделювання фізичних систем. Modelica надає високорівневі описи гібридних динамічних систем, які використовуються як основа для різних видів обчислень в JModelica.org, включаючи моделювання, аналіз чутливості та оптимізацію.

Проблеми динамічної оптимізації, включаючи оптимальне керування, оптимізацію траєкторії, оптимізацію параметрів і калібрування моделі, можна сформулювати і вирішити за допомогою JModelica.org. Розширення Optimica дозволяє на високому рівні формулювати динамічні задачі оптимізації на основі моделей Modelica.

JModelica.org сумісний зі стандартним інтерфейсом функціонального макету (FMI) та функціональними макетами (FMU), створеними JModelica.org або іншим FMI-сумісним інструментом, які можна змоделювати в середовищі Python.

Незалежне порівняння між JModelica.org і системами оптимізації ACADO Toolkit, IPOPT і CppAD, наведено в звіті з відкритим вихідним кодом для нелінійної обмеженої оптимізації динамічних систем.

2.3.3 Платформа для моделювання Wolfram SystemModeler

Wolfram SystemModeler, розроблений компанією Wolfram Research, є платформою для математичного та комп'ютерного моделювання фізико-технічних та біохімічних об'єктів і систем на базі мови Modelica. Він представляє інтерактивне графічне середовище для математичного та комп'ютерного моделювання, а також безліч конфігуруючих бібліотек. [15]

Серед функціональних можливостей Wolfram SystemModeler можна відмітити:

- використання за основу незапатентованої, об'єктно-орієнтованої мови Modelica, що базується на диференціальних рівняннях.
- графічний користувальницький інтерфейс для drag-and-drop моделювання

- текстовий редактор для програмування на мові Modelica за допомогою рівнянь, симуляцій, документування та аналізу
- компонентно-орієнтоване і блочно-орієнтоване моделювання.

2.3.4 Комерційне середовище для моделювання Dymola

Dymola - це комерційне середовище моделювання, засноване на відкритій мові моделювання Modelica. Dymola (Dynamic Modeling Laboratory) - це інструмент для моделювання інтегрованих і комплексних систем в автомобілях, аерокосмічній промисловості, робототехніці та інших галузях. [16]

Великі і складні системи складаються з компонентних моделей; Математичні рівняння описують динамічну поведінку системи. Розроблена європейською компанією Dassault Systèmes, Dymola доступна як окремий продукт і інтегрована в 3DEXPERIENCE як частина CATIA.

Dymola має мульти інженерні можливості, які означають, що моделі можуть складатися з компонентів багатьох інженерних областей. Використовуючи мову Modelica, підсистеми представлені взаємопов'язаними компонентами; на найнижчому рівні динамічна поведінка описується математичними рівняннями або алгоритмами. З'єднання між компонентами утворюють додаткові рівняння. Dymola обробляє повну систему рівнянь для створення ефективного коду моделювання.

Вузькоспеціалізовані частини представлені бібліотеками Modelica, що містять компоненти для механічних, електричних, керуючих, теплових, пневматичних, гідравлічних, силових, термодинамічних, динамічних тощо.

2.3.5 Середовище для моделювання OpenModelica

OpenModelica є безкоштовним і open source середовищем, заснованим на мові Modelica і призначеним для моделювання, симулювання, оптимізації та аналізу складних динамічних систем. Це програмне забезпечення активно

розвивається. Open Source Modelica Consortium є спільним проектом RISE SICS East AB і Лінчепінгского університетом (Linköping University).

OpenModelica використовується в наукових цілях і на виробництві. У промисловості використовується в області оптимізації енергопостачання, автомобілебудуванні та водоочищенні.

OpenModelica включає готові, стандартні блоки:

- механічні
- електричні
- електроні
- електродвигуни
- гідравліку
- термодинаміку

За своїми можливостями OpenModelica наближається до таких середовищ як Matlab Simulink, Scilab xCos, маючи при цьому значно більш зручне представлення моделі і можливість кастомізації готових компонентів.

Є можливість компілювати код блоків для подальшого впровадження в Matlab і Scilab xCos. Також OpenModelica повністю сумісна з усіма бібліотеками Modelica.

OpenModelica Compiler (OMC) є компілятором Modelica, і в свою чергу транслює код Modelica в код мови C, включаючи класи, функції та змінні з користувацьких моделей або бібліотек. Компілятор також включає інтерпретатор Modelica для інтерактивного використання і обчислення виразів.

Висновки до розділу 2

В даному розділі було розглянуто ряд програмних засобів для моделювання АСК. В першу чергу, було розглянуто мову моделювання SysML. SysML як розширена версія відкритого стандарту UML відкриває ряд нових можливостей. SysML цікавий можливістю генерувати код з структурних діаграм моделей. Також, на етапі проектування АСК важливу роль відіграють

діаграми вимог та параметричні діаграми, які приносять чітке розуміння суті моделі, та взаємозвязків між різними об’єктами в системі. Також увагу привернув відкритий стандарт AutomationML, з його можливістю будувати чітко описані моделі об’єктів та подальшим формуванням XML даних які можна використати в зовсім інших проектах. Проте, при виборі засобу моделювання було одназначно вирішено використовувати об’єктно-орієнтовану мову моделювання Modelica. Серед переваг моделіки варто зазначити велику кількість задокументованих матеріалів, котрі допомагають освоїти даний інструмент. Першим інструментом серед Modelica-залежних мов був розглянутий інструмент Dymola. Та нажаль, майже одразі довелесь відмовитися, даний інструмент не підходив під вхідні вимоги, адже він комерційний, а демо версія не може продемонструвати всі переваги. Надалі було розглянуто UML плагін для IDE Eclipse – ModelicaML. Інструмент сам по собі є дуже цікавим. Для відображення UI частини він використовує Eclipse Papyrus Modeling environment. Сам механізм формування коду с графічної моделі реалізований на базі Acceleo. В ході дослідження ModelicaML його не вдалося навіть інсталювати. Адже в залежностях для нього вказаний Acceleo версії 2.8, а на момент 2019 року, найстаріша доступна версія Acceleo – 3.0. Таким чином, можна констатувати, що на момент 2019 року ModelicaML є застарілим, недієздатним інструментом. Далі було оглянуто три продукти на базі мови Modelica: JModelica.org, Wolfram SystemModeler та OpenModelica. Перший з них, а саме JModelica.org цікавий загальною інтеграцією з високорівневою мовою програмування Python. Завдяки цьому на базі однієї моделі можна одночасно використовувати всі переваги мов Modelica та Python. Варіант з Wolfram SystemModeler також був відхилений, адже даний продукт платний. Отже вибір припав на OpenModelica. Серед переваг варто відмітити, що це продукт з відкритим кодом, велику бібліотеку стандартних класів та просту можливість змінювати та створювавати нові класи-модулі. Отже, надалі для моделювання АСК буде використовуватися OpenModelica.

3 ОПИС СТРУКТУРНОЇ СХЕМИ АСК

3.1 АСК з гідродинамічним процесом

Структурна схема гідродинамічного процесу в АСК зображена в «Схема електрична структурна Е1». АСК складається з трьох резервуарів для зберігання рідин, чотирьох дискретних клапанів, контролера для керування системою та трьох кнопок для взаємодії оператора з системою. Дана структурна схема демонструє систему автоматичної очистки рідини від дрібних домішок відстоюванням. На вхід система отримує рідину з домішками, на вихід повертає очищену рідину та рідину з високою концентрацією домішок.

В ході виконання дипломного проекту буде побудована модель системи автоматичного керування процесом відстоювання рідини.

Дана система має три режима роботи. Перший режим – нормальний робочий цикл системи. Нормальний робочий цикл системи виглядає приблизно так.

В початковий момент часу t=0 секунд:

1. кнопка «btnStart» не нажата;
2. кнопка «btnStop» не нажата
3. кнопка «btnBreakWork» не нажата
4. клапан «valve1» закритий
5. клапан «valve2» закритий
6. клапан «valve3» закритий
7. клапан «valve4» закритий
8. резервуар «tank1» пустий
9. резервуар «tank2» пустий
10. резервуар «tank3» пустий,
11. час очікування відстоювання в таймері «fallingTime» встановлено 60 секунд

12. відсоток очікуваних відходів для резервуару «tank1» встановлено 5%

13. відсоток максимального наповнення резервуару «tank1» встановлено 90%

14. відсоток максимального наповнення резервуару «tank2» встановлено 95%

В момент часу t=20 секунд оператор натискає кнопку «btnStart». Кнопка «btnStart» нажата («btnStart.on») після чого розпочинається робочий цикл системи:

1. клапан «valve1» відкрито

2. рідина подається в систему в резервуар «tank1», резервуар починає наповнюватися

Коли рівень рідини в резервуарі «tank1» досягає максимального значення(«tank1.level» > «90% * tank1. height»):

1. клапан «valve1» закрито

2. запускається таймер «fallingTime»

Після завершення роботи таймера «fallingTime»:

1. клапан «valve2» відкрито

2. рідина з резервуара «tank1» перетікає в резервуар «tank2»

Коли рівень рідини в резервуарі «tank1» досягає мінімального значення тобто коли в резервуарі «tank1» залишається тільки осад («tank1.level» < «5% * tank1. height»), або коли рівень рідини в резервуарі «tank2» досягає максимального значення тобто коли резервуар «tank2» переповнюється («tank2.level» > «95% * tank2. height»),:

1. клапан «valve2» закривається

2. клапан «valve3» відкривається

3. очищена рідина з резервуара «tank2» йде на вихід з системи «outcome»

Коли рівень рідини в резервуарі «tank2» досягає мінімального значення тобто коли резервуар «tank2» залишається пустим («tank2.level» < «0.01»):

1. клапан «valve3» закривається
2. клапан «valve4» відкривається
3. залишки осаду з резервуара «tank1» перетікають в резервуар для збору відходів «tank3»

Коли рівень рідини в резервуарі «tank1» досягає мінімального значення тобто коли резервуар «tank1» залишається пустим («tank1.level» < «0.01») - клапан «valve4» закривається.

Якщо кнопка «btnStart» досі нажата («btnStart.on») робочий цикл виконується спочатку.

Другий режим роботи – режим очікування. Коли оператор натискає кнопку «btnStop»:

1. клапан «valve1» закритий
2. клапан «valve2» закритий
3. клапан «valve3» закритий
4. клапан «valve4» закритий
5. кнопка «btnStart» не нажата;
6. кнопка «btnBreakWork» не нажата

Система перебуває в режимі очікування поки оператор не натисне кнопку «btnStart» або «btnBreakWork».

І останній режим – режим екстренного зливу очищеної рідини. Коли оператор натискає кнопку «btnBreakWork»:

1. клапан «valve1» закритий
2. клапан «valve2» закритий
3. клапан «valve3» відкритий
4. клапан «valve4» закритий
5. кнопка «btnStart» не нажата;
6. кнопка «btnStopt» не нажата

Коли рівень рідини в резервуарі «tank2» досягає мінімального значення тобто коли резервуар «tank2» залишається пустим («tank2.level» < «0.01») - клапан «valve3» закривається і система переходить в режим очікування.

Висновки до розділу 3

В цьому розділі була описана типова гідродинамічна АСК. Були описані її режими роботи, та модель поведінки. В наступному розділі буде реалізована модель даної АСК засобами OpenModelica, та проведені симуляції роботи даної схеми.

4 ІМІТАЦІЙНЕ МОДЕЛЮВАННЯ ЗАСОБАМИ OPENMODELICA

4.1 Імітаційне моделювання електричної схеми

Робота з OpenModelica слід розпочати з простої імітаційної моделі. Мета даного розділу – змоделювати роботу ідеального діода та отримати його ВАХ.

Структурна схема моделі з діодом складається з джерела струму, діода, резистора та заземлення.

В якості джерела струму використано компонент Modelica.Electrical.Analog.Sources.SineVoltage. Даний компонент обладнаний двома портами: катодом і анодом. При налаштуванні компонента встановимо значення напруги - 10 В, а значення частоти – 1 Гц.

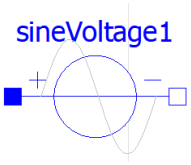

Рисунок 4.1 – Зовнішній вигляд компонента Modelica.Electrical.Analog.Sources.SineVoltage

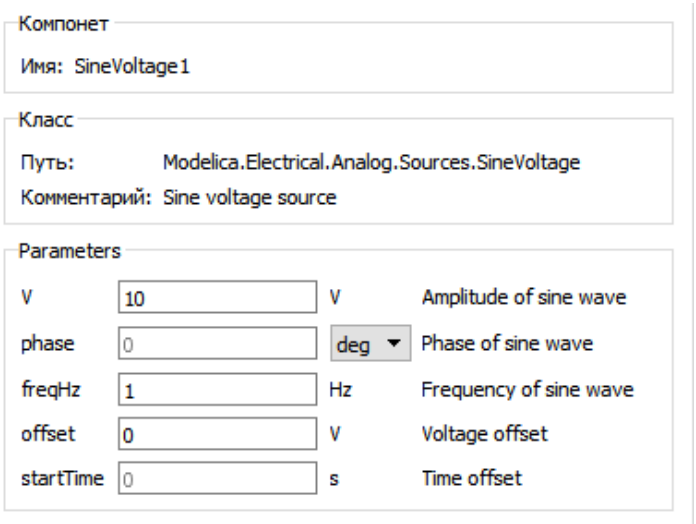

Рисунок 4.2 – Налаштування параметрів компонента Modelica.Electrical.Analog.Sources.SineVoltage SineVoltage1

Як резистор використано компонент Modelica.Electrical.Analog.Basic.Resistor. Даний компонент обладнаний трьома портами: температурним портом, входом та виходом. При налаштуванні компонента встановимо значення опору - 2 Ом. Температурними параметрами резистора в даній моделі нехтується.

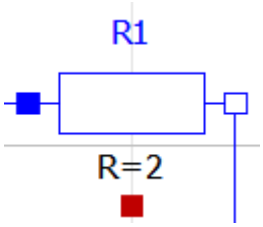

Рисунок 4.3 – Зовнішній вигляд компонента Modelica.Electrical.Analog. Resistor

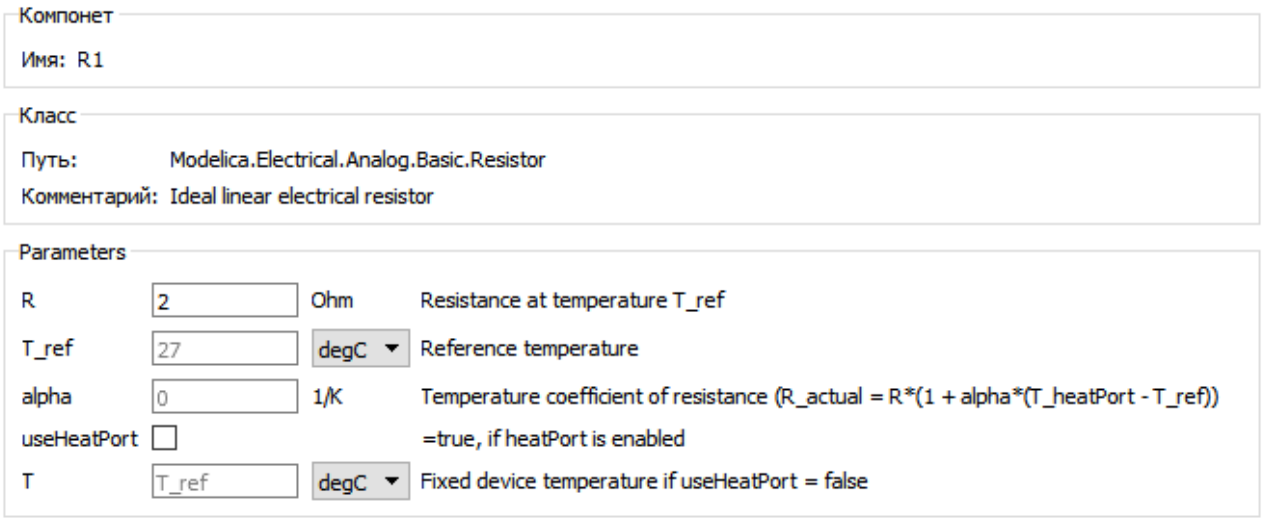

Рисунок 4.4 – Налаштування параметрів компонента Modelica.Electrical.Analog. Resistor R1

Для моделювання діода використано компонент Modelica.Electrical.Analog.Ideal.IdealDiode. Даний компонент обладнаний трьома портами: температурним портом, анодом і катодом. При налаштуванні компонента встановимо значення диференційного опору – 0 Ом. Температурними параметрами резистора в даній моделі нехтується.

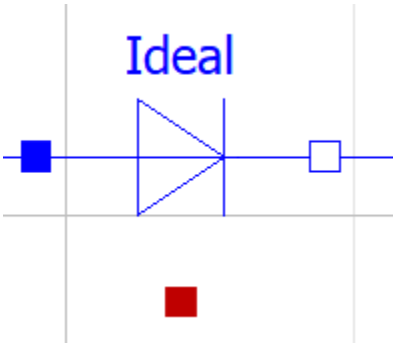

Рисунок 4.5 – Зовнішній вигляд компонента Modelica.Electrical.Analog.Ideal.IdealDiode

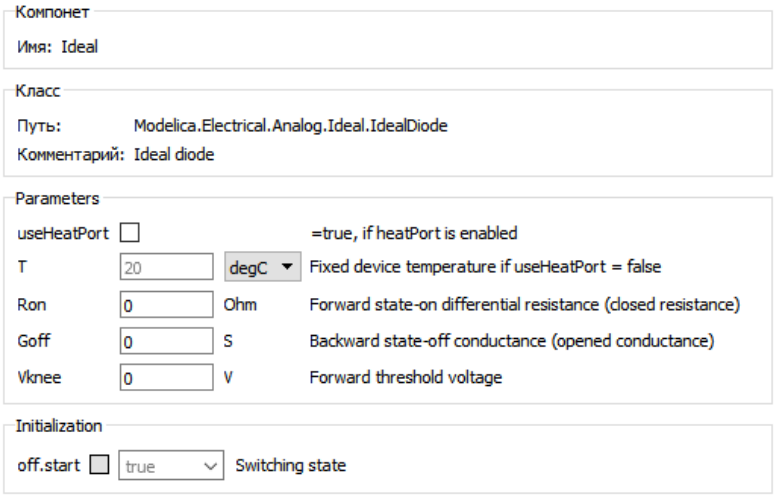

Рисунок 4.6 – Налаштування параметрів компонента Modelica.Electrical.Analog.Ideal.IdealDiode Ideal

Для моделювання заземлення в електричному колі використано компонент Modelica.Electrical.Analog.Basic.Ground.

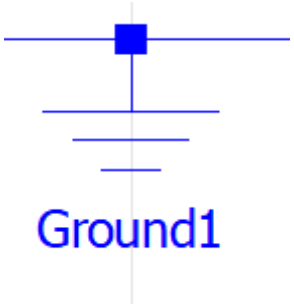

Рисунок 4.7 – Зовнішній вигляд компонента Modelica.Electrical.Analog.Basic.Ground

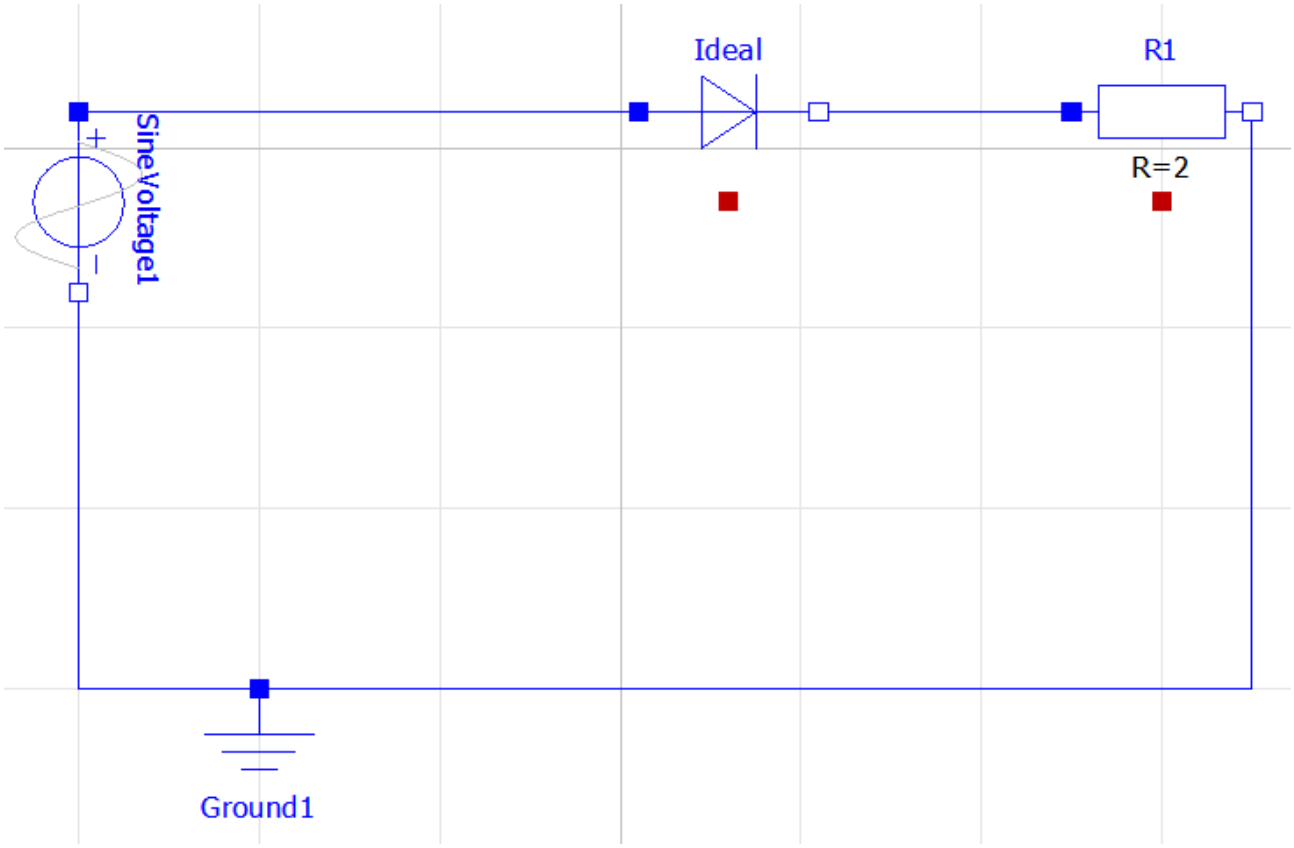

Рисунок 4.8 – Модель електронної схеми

Далі слід перевірити дієздатнійсть даної моделі засобами OpenModelica. Для цього на панелі «Симуляції» слід натиснути «Перевірити модель».

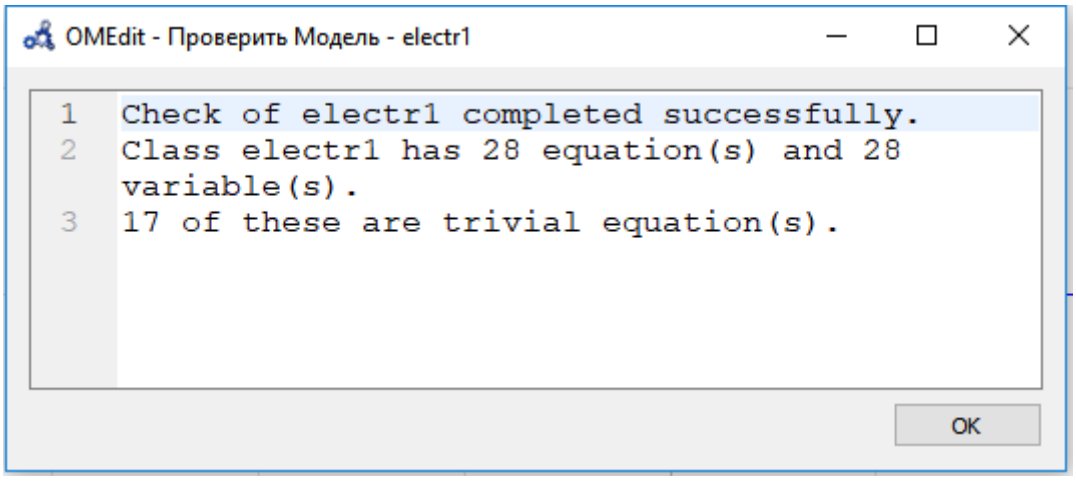

Рисунок 4.9 – Результат перевірки моделі електричної схеми

Після цього проведемо симуляцію роботи, натиснути «Симуляція» потім «Налаштування симуляції». Та встановити час симуляції від 0 секунд до 1 секунд.

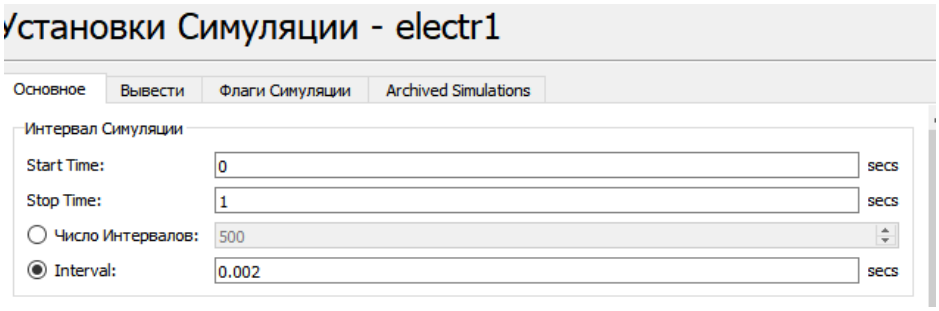

Рисунок 4.10 – Налаштування симуляції

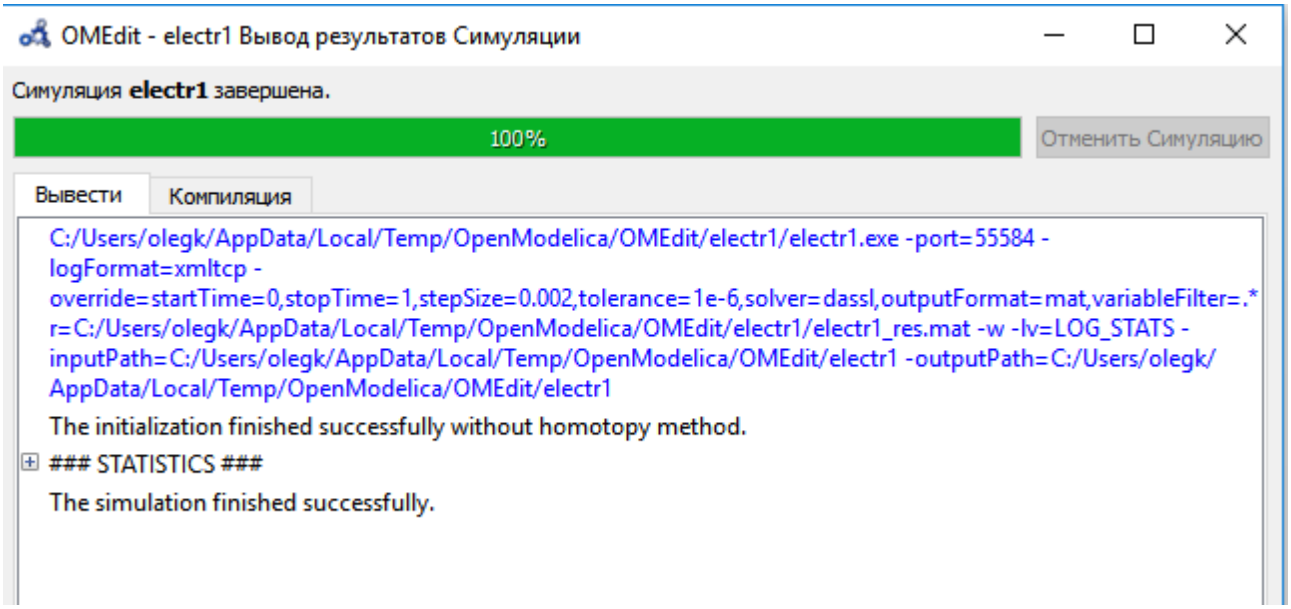

Рисунок 4.11 – Результат компіляції моделі

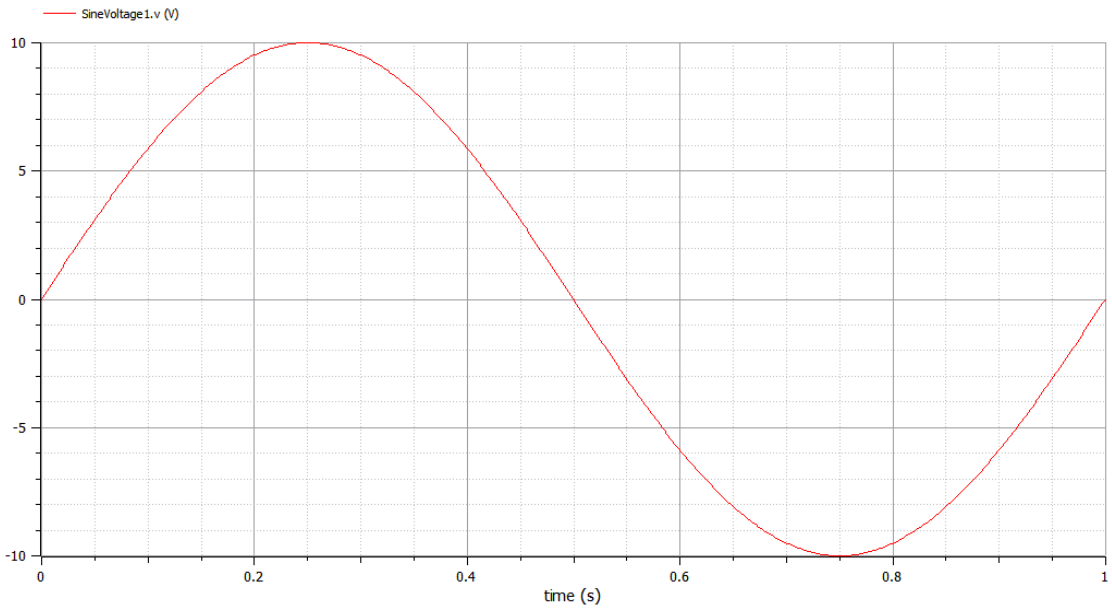

Рисунок 4.12 – Графік залежності напруги джерела від часу

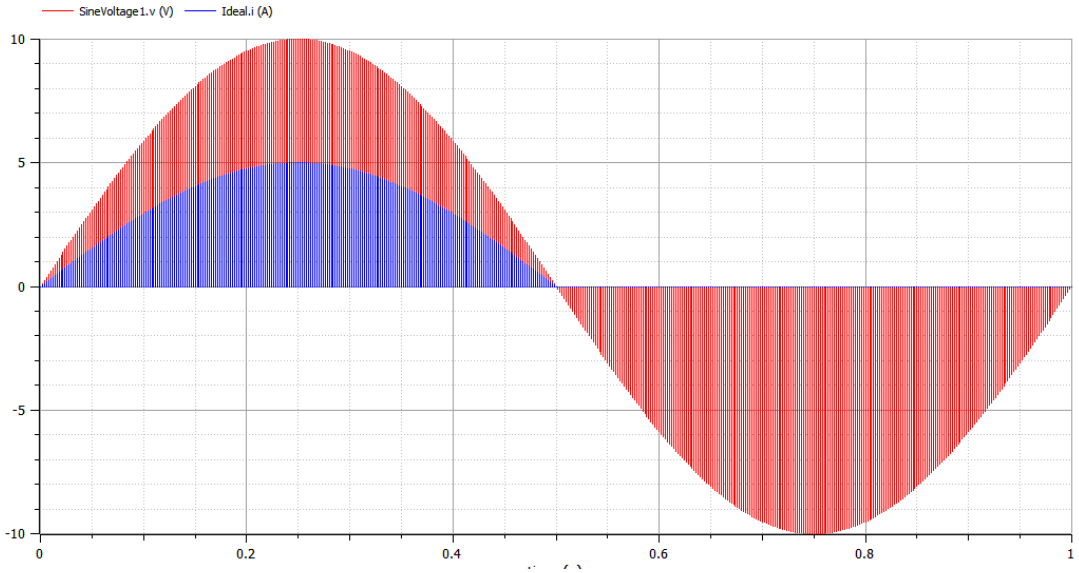

Рисунок 4.13 – Графік залежності напруги та струму в схемі від часу

4.2 Імітаційне моделювання механічного процесу

Наступним кроком знайомства з OpenModelica буде розглянуто модуль «Mechanics». Мета даного розділу – змоделювати пружинний маятник.та отримати анімацію роботи моделі.

Структурна схема моделі складається з системи координат, пружини та вантажу.

Modelica.Mechanics.MultiBody.World – глобальна система координат. В налаштування компонента слід встановити «animateGravity» в положення «true».

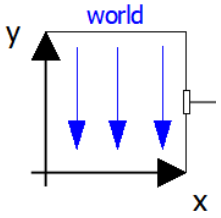

Рисунок 4.14 – Зовнішній вигляд компонента Modelica.Mechanics.MultiBody.World

Modelica.Mechanics.MultiBody.Forces.Spring – пружина. В налаштування компонента слід встановити сила коефіцієнт жорсткості – 40 Н/м. Довжина пружини в нормальному стані – 0.2 м. Маса пружини – 0.5 кг.

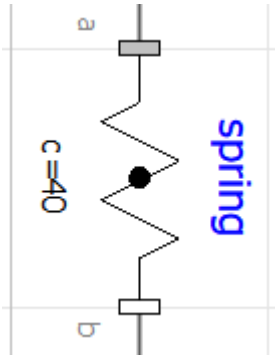

Рисунок 4.15 – Зовнішній вигляд компонента Modelica.Mechanics.MultiBody.Forces.Spring

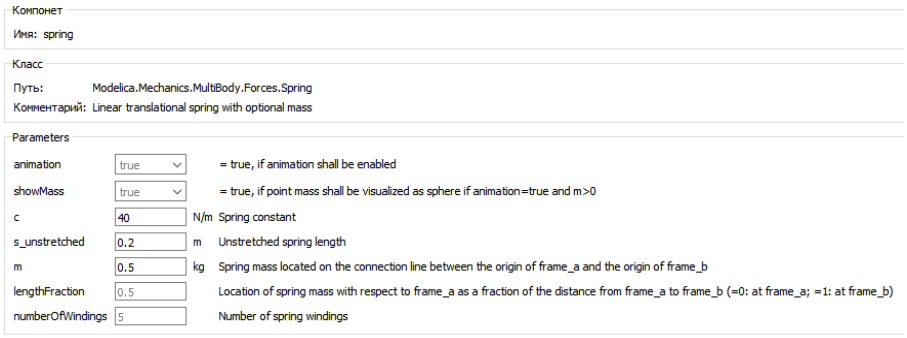

Рисунок 4.16 – Налаштування компонента Modelica.Mechanics.MultiBody.Forces.Spring spring

Модель вантажу імітує компонент - Modelica.Mechanics.MultiBody.Parts.Body. В налаштуваннях компонента встановимо масу вантажу – 1 кг.

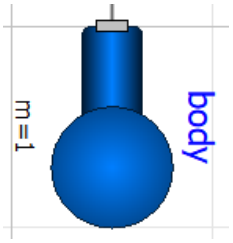

Рисунок 4.17 – Зовнішній вигляд компонента Modelica.Mechanics.MultiBody.Parts.Body body

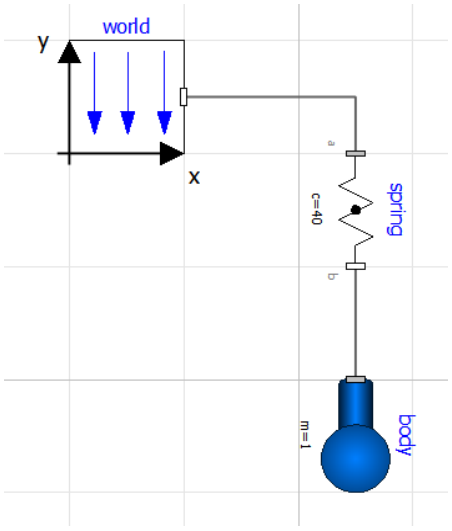

Рисунок 4.18 – Модель механічного процесу

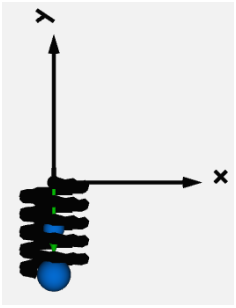

Рисунок 4.19 – Анімована модель пружинного маятника в момент часу t=0 секунд

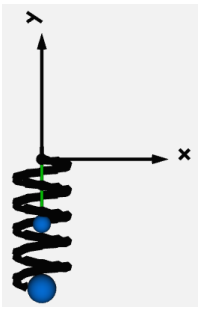

Рисунок 4.20 – Анімована модель пружинного маятника в момент часу t=0.8 секунд

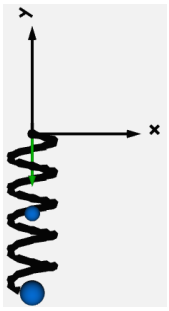

Рисунок 4.21 – Анімована модель пружинного маятника в момент часу t=1.6 секунд

4.3 Моделювання теплотехнічного процесу

Далі засобами OpenModelica буде побудована нескладна модель теплотехнічного процесу засобами модулю «Thermal».

Структурна схема даної моделі складається з двох накопичувальних баків, насосу, нагрівача, труби та конденсатора тепла.

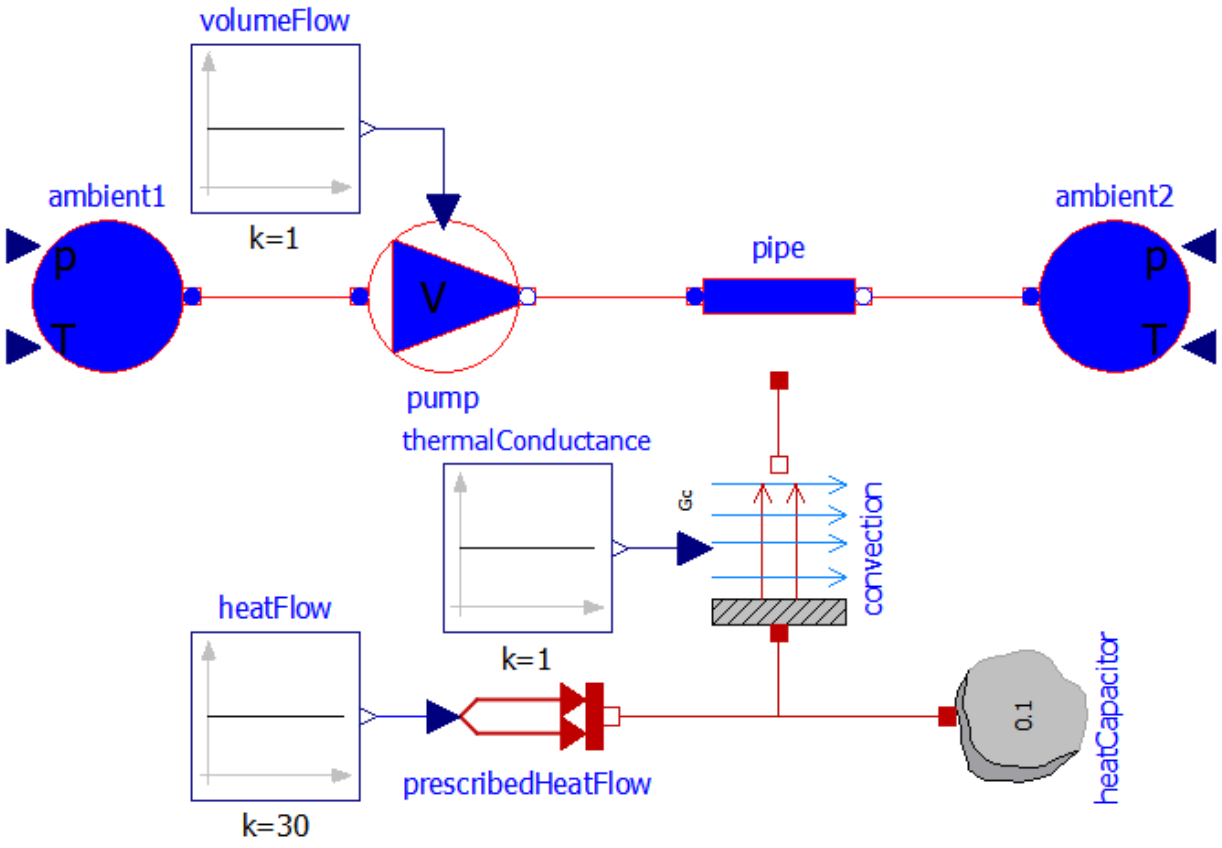

Рисунок 4.22 – Структурна схема теплотехнічного процесу

В даній схемі використано два джерела рідини: ambient1, ambient2. Помпа - pump, яка перекачує рідину з першого джерела в друге через трубу - pipe. Нагрівач – prescribedHeatFlow. Конденсатор тепла, який складається з heatCapacitor та convection елемента.

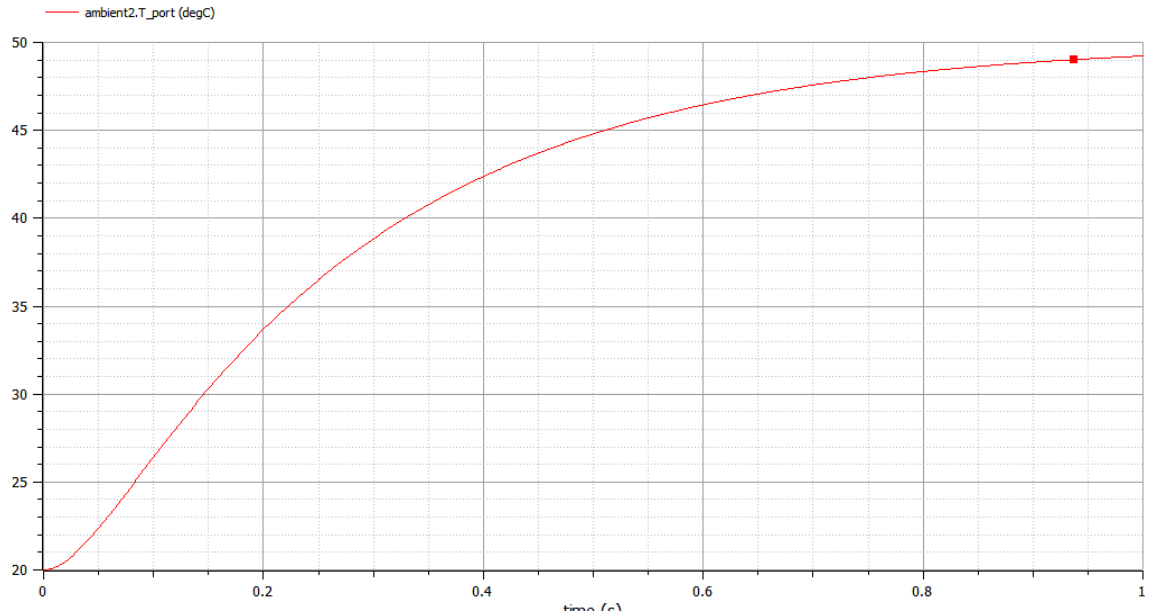

Рисунок 4.23 – Графік залежності температури рідини в другому джерелі від часу

4.4 Моделювання АСК з гідродинамічним процесом

Кожна гідродинамічна модель розпочинається з класу Modelica.Fluid.System - системний компонент необхідний для будь-якої гідродинамічної моделі для забезпечення загальносистемних параметрів, таких як умови навколишнього середовища та загальні припущення щодо моделювання. Параметри системи передаються до всіх класів які наслідують Modelica.Fluid з використанням внутрішнього механізму (автоматичного) або зовнішнього механізму (перевизначення – redeclare).

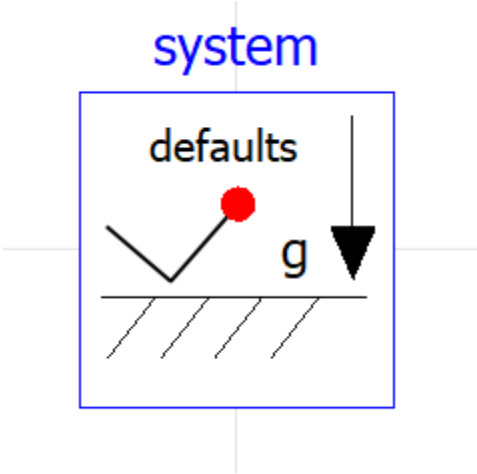

Рисунок 4.24 – Зовнішній вигляд компонента Modelica.Fluid.System

Для даної моделі встановимо значення температури навколишнього середовища 20 градусів по шкалі Цельсія. Значення атмосферного тиску – 101325 Па. А значення прискорення вільного падіння – 9.8 м/с^2 .

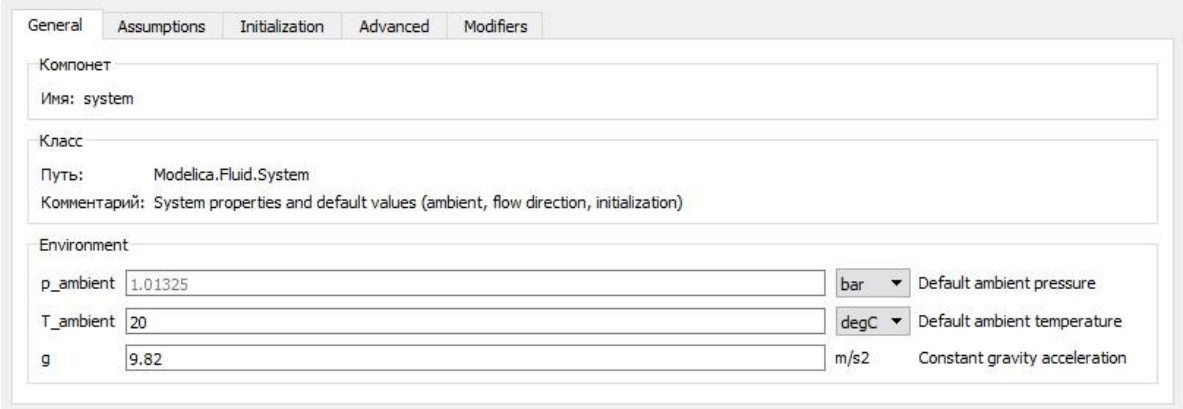

Рисунок 4.25 – Налаштування параметрів компонента Modelica.Fluid.System system

Наступним кроком слід задати загальні параметри рідини для системи. Для спрощення математичної моделі, будемо вважати, що рідина з домішками поводить себе аналогічно до звичайної води. Отже в якості рідини буде використана вода. Для цього слід оголосити глобальну змінну:

```
package Medium = Modelica.Media.Water.ConstantPropertyLiquidWater;
```

Після чого, слід памятати про необхідність перевизначення атрибута «Medium» в конструкторах класів для всіх об'єктів які наслідують Modelica.Fluid.

Також слід описати вхід та вихід для системи. Запропонована модель на вхід та на вихід використовує незалежні від моделі джерела. В якості вхідного джерела рідини використаємо компенет класу Modelica.Fluid.Sources.Boundary_pT source. А в якості виходу для рідини з системи використаємо компенет класу Modelica.Fluid.Sources.Boundary_pT outcome. В інтерфейсі даного компонента є 4 параметричних входа, які приймають числа з плаваючою точкою: p_in – тиск рідини, T_in – температура рідини, X_in – склад рідини, C_in – концентрація рідини та 3 порти виходу.

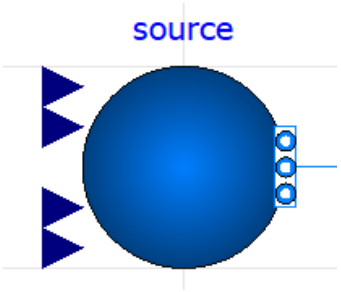

Рисунок 4.26 – Зовнішній вигляд компонента Modelica.Fluid.Sources.Boundary_pT

Спочатку задамо всі параметри для вхідного джерела рідини. Нехай значення тиску з яким рідина подається на вхід – 2.5 МПа, тобто майже в 2.5 рази більше за значення атмосферного тиску. Величина температури поданої рідини рівна температурі навколишнього середовища – 20 градусів по шкалі Цельсія. Кількість портів – 1, адже даний компонент з’єднаний тільки з клапаном valve1.

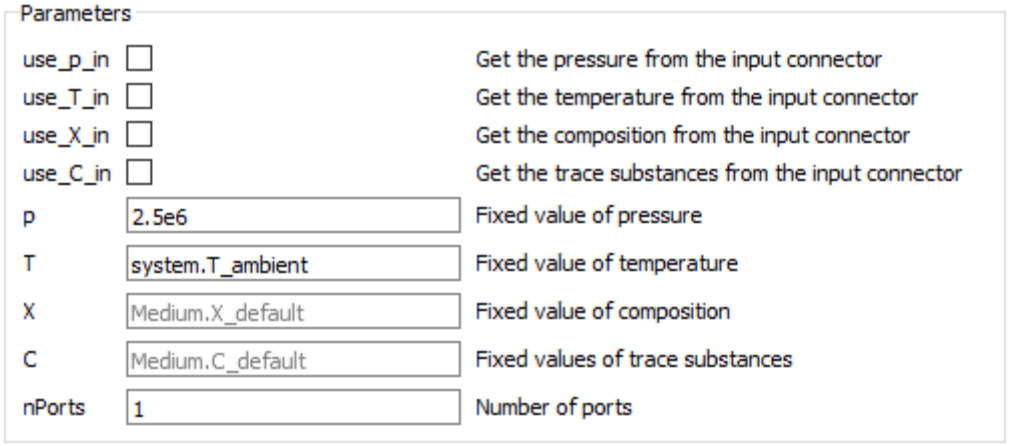

Рисунок 4.27 – Налаштування параметрів компонента Modelica.Fluid.Sources.Boundary_pT source

Далі задамо всі необхідні параметри для вихідного джерела рідини. Будемо вважати, що рідина на вихід системи подається з номальним атмосферним тиском. Температура рідини вихідної рідини рівна температурі навколишнього середовища – 20 градусів по шкалі Цельсія. Кількість портів – 1, адже даний компонент з’єднаний тільки з клапаном valve3.

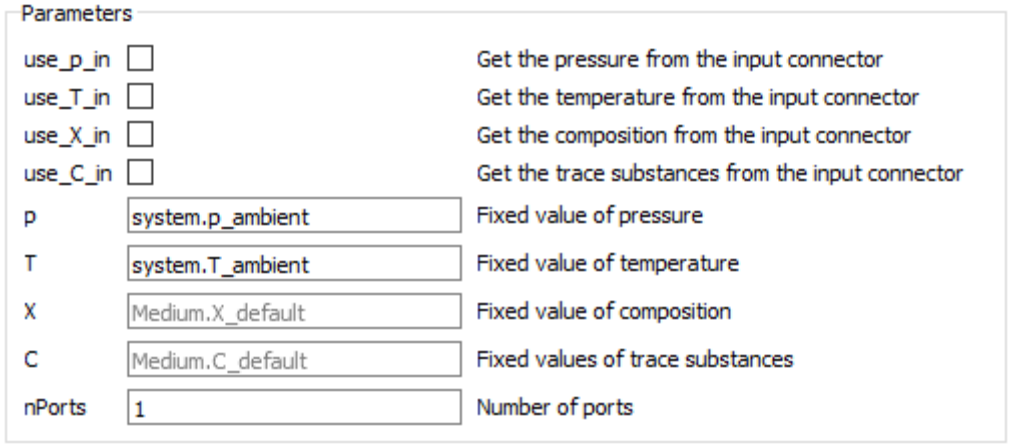

Рисунок 4.28 – Налаштування параметрів компонента Modelica.Fluid.Sources.Boundary_pT outcome

Також в АСК використано три резервуара для рідини. Клас Modelica.Fluid.Vessels.OpenTank - модель резервуара, відкритого для навколишнього середовища при фіксованому тиску навколишнього середовища «p_ambient» заданого компонентом system. Величина з'єднувальних портів

являє собою отвір для рідин при конфігурованих висотах, відносно дна бака. Рідина може витікати або затікати в кожен з портів. Даний компонент також має 1 температурний порт для можливості нагрівання/охолодження рідини всередині та 3 порти-отвори для рідини.

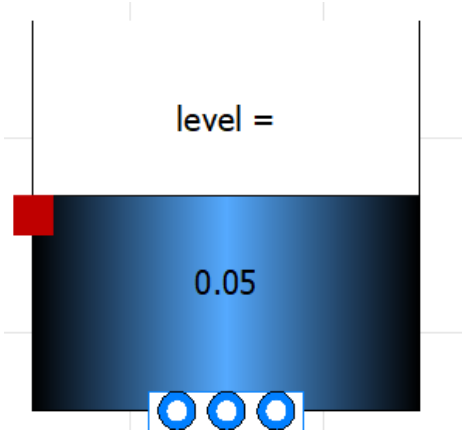

Рисунок 4.29 – Зовнішній вигляд компонента Modelica.Fluid.Vessels.OpenTank

Для даного компонента існують такі правила:

- Даний резервуар має циліндричну форму.
- Резервуар наповнюється однією або кількома речовинами, які мають густину, що перевищує густину навколишнього середовища.
- Рідина має рівномірну густину, температуру і масову частку.
- Жодна рідина не переливається з бака через відкритий верх (моделювання припиняється з помилкою, якщо рівень рідини перевищує по висоті задану висоту резервуара.
- Початкове значення рівня рідини в резервуарі ненульове.

В якості резервуару для відстоювання використано Modelica.Fluid.Vessels.OpenTank tank1. Надалі задамо параметри для першого резервуару. Висота резервуару – 6 м, площа резервуару – 5 м^2 . Початкове значення рівня рідини в резервуарі – 0.05 м. Кількість портів – 3, адже в даний резервуар рідина потрапляє від джерела через клапан valve1, від цього резервуару через клапан valve2 до резервуару для зберігання очищеної рідини,

також від цього резервуару через клапан valve4 залишки осаду потрапляють в резервуар для збору відходів.

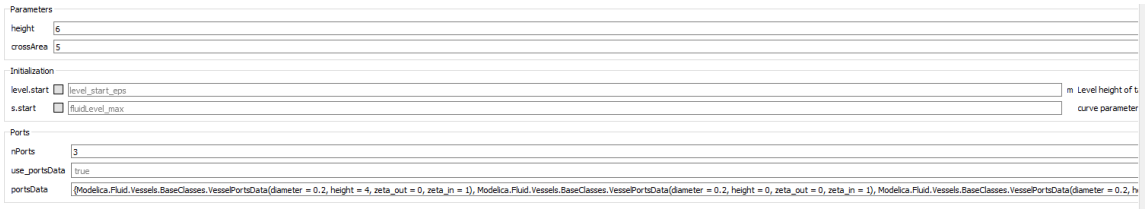

Рисунок 4.30 – Налаштування параметрів компонента Modelica.Fluid.Vessels.OpenTank tank1

В якості резервуару для збору очищеної рідини використано Modelica.Fluid.Vessels.OpenTank tank2. Надалі задамо параметри для першого резервуару. Висота резервуару – 25 м, площа резервуару – 5 м^2 . Початкове значення рівня рідини в резервуарі – 0.05 м. Кількість портів – 2, адже в даний резервуар рідина потрапляє після відстоювання через клапан valve2, також від цього резервуару через клапан valve3 очищена рідина потрапляю на вихід з системи.

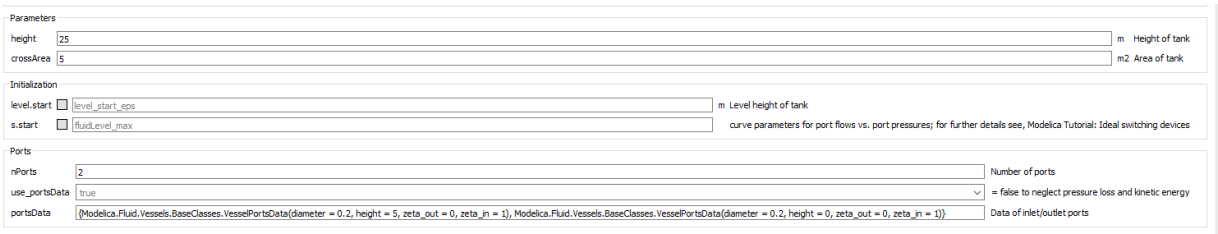

Рисунок 4.31 – Налаштування параметрів компонента Modelica.Fluid.Vessels.OpenTank tank2

В якості резервуару для збору очищеної рідини використано Modelica.Fluid.Vessels.OpenTank tank3. Надалі задамо параметри для першого резервуару. Висота резервуару – 6 м, площа резервуару – 5 м^2 . Початкове значення рівня рідини в резервуарі – 0.05 м. Кількість портів – 1, адже в даний резервуар залишки осаду потрапляють після відстоювання в резервуарі tank1 через клапан valve4.

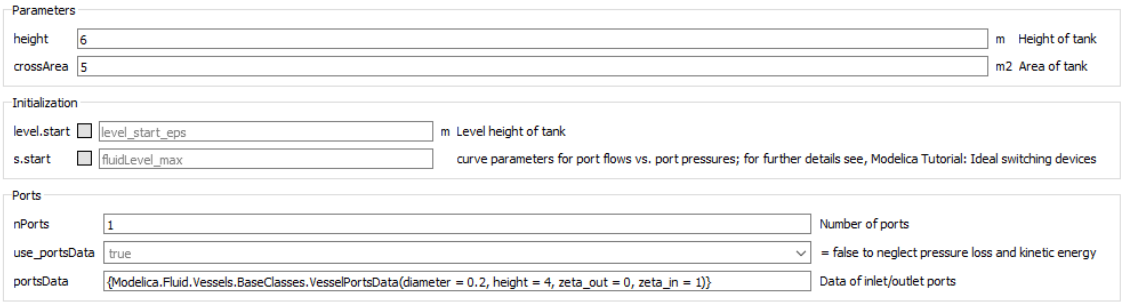

Рисунок 4.32 – Налаштування параметрів компонента Modelica.Fluid.Vessels.OpenTank tank3

Далі додамо моделі клапанів в систему. Клас Modelica.Fluid.Valves.ValveDiscrete – це дуже проста модель клапана, яка забезпечує малий перепад тиску, який пропорційний потоку рідини, якщо булевий сигнал поданий на вхід компонента є істиною. Ця модель може бути використана для спрощеного моделювання запобіжних клапанів, коли не важливо точно описувати втрати тиску, при відкритті клапана. Модель є адіабатичною (відсутня втрата тепла в навколишнє середовище) і нехтує змінами кінетичної енергії між входом та виходом клапана. Цей пристрій обладнано одним булевим входом, який відповідає за відкривання та закривання клапану, одним вхідним портом в який подається вода та обдним вихідним портом, з якого рідина витікає.

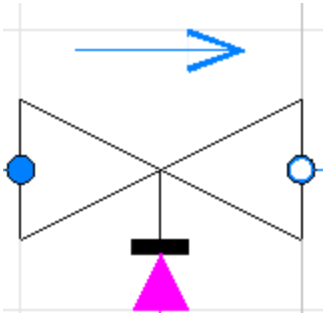

Рисунок 4.33 – Зовнішній вигляд компонента Modelica.Fluid.Valves.ValveDiscrete

В даній моделі використано 4 дискретних клапана. Перший клапан Modelica.Fluid.Valves.ValveDiscrete valve1 з'єднує вхідне джерело рідини та резервуар для відстоювання tank1.

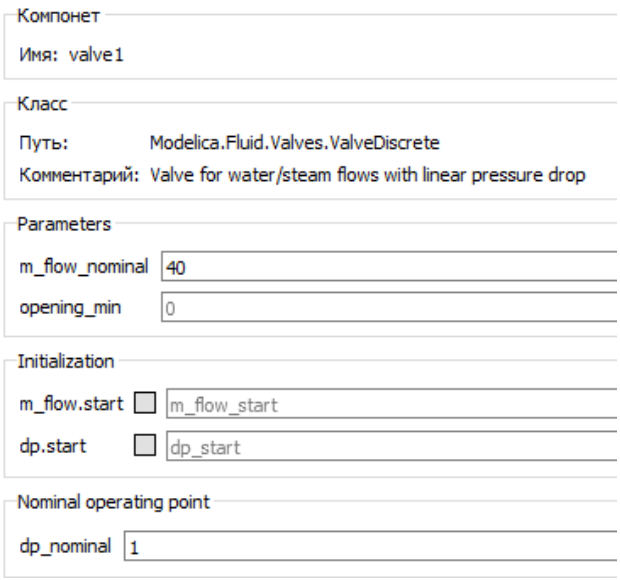

Рисунок 4.34 – Налаштування параметрів компонента Modelica.Fluid.Valves.ValveDiscrete valve1

Другий клапан Modelica.Fluid.Valves.ValveDiscrete valve2 з'єднує резервуар для відстоювання tank1 та резервуар для збору очищеної рідини tank2.

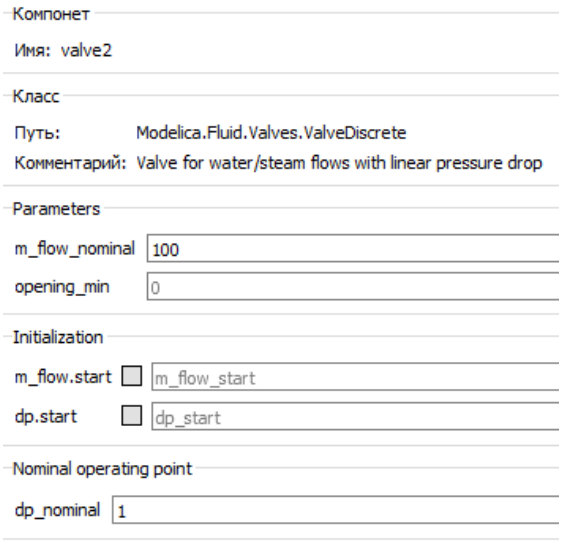

Рисунок 4.35 – Налаштування параметрів компонента Modelica.Fluid.Valves.ValveDiscrete valve2

Третій клапан Modelica.Fluid.Valves.ValveDiscrete valve3 з’єднує резервуар для збору очищеної рідини tank2 та outcome.

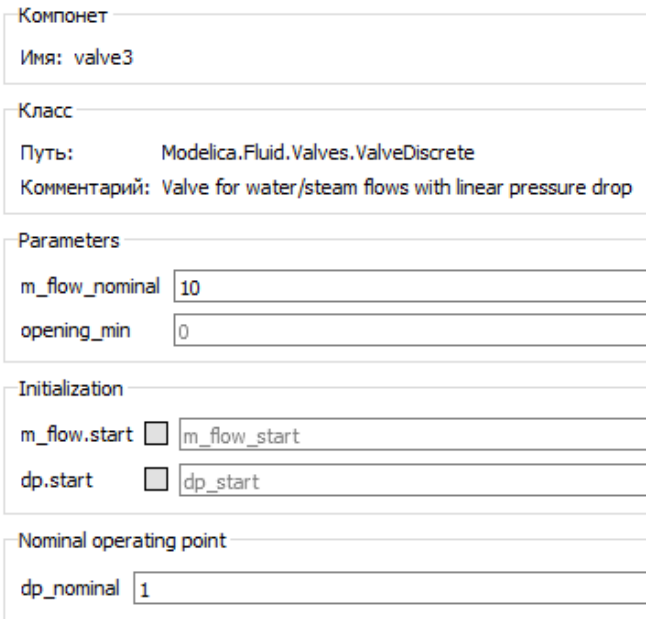

Рисунок 4.36 – Налаштування параметрів компонента Modelica.Fluid.Valves.ValveDiscrete valve3

Четвертий клапан Modelica.Fluid.Valves.ValveDiscrete valve4 з’єднує резервуар для відстоювання рідини tank1 та резервуар для збору осаду tank3.

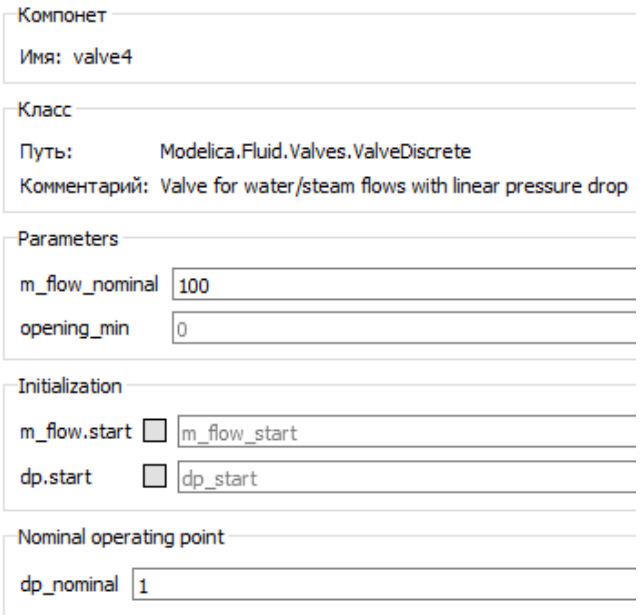

Рисунок 4.37 – Налаштування параметрів компонента Modelica.Fluid.Valves.ValveDiscrete valve4

Оператор взаємодіє з даною АСК за допомогою кнопок. Дана схема АСК обладнана трьома кнопками-перемикачами режимів роботи. Для імітаційної роботи цих кнопок використаємо компонент Modelica.StateGraph.Temporary.RadioButton btnStart. Даний клас представляє з себе типовий radiobutton, який являє собою кнопку, яка встановлює вихідний сигнал у булеве знаення true, коли натискається, і скидається в булеве значення false, коли виконується умова задана в «reset». Отже в схему додано три кнопки: btnStart, btnStop, btnBreakWork.

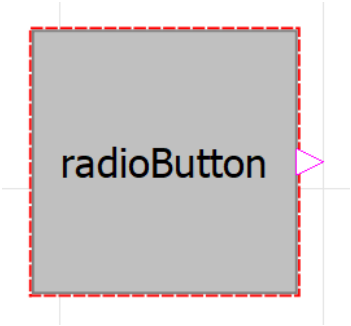

Рисунок 4.38 – Зовнішній вигляд компонента Modelica.StateGraph.Temporary.RadioButton

Кнопка btnStart – при натисканні запускає нормальний робочий цикл системи. Для імітаційної моделі задамо момент натискання даної кнопки через 20 секунд після запуску симуляції. В умови скидування задаємо btnStop.on та btnBreakWork.on, тобто кнопка btnStart вимикається відразу після натискання однієї з цих кнопок.

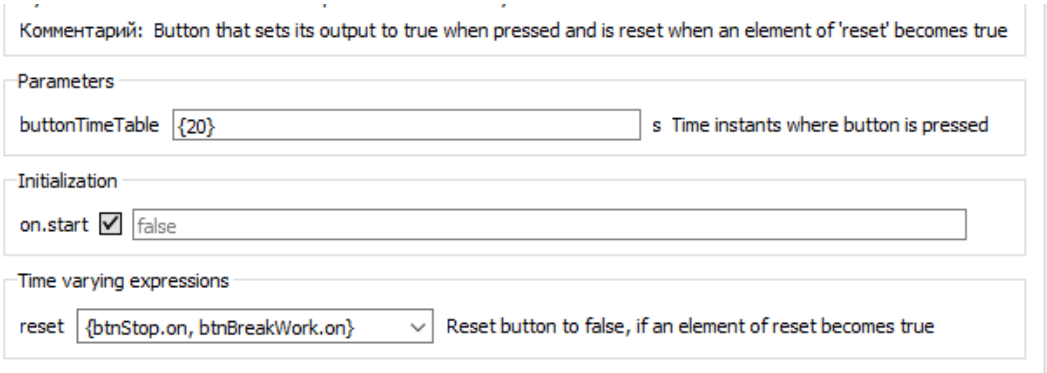

Рисунок 4.39 – Налаштування параметрів компонента Modelica.StateGraph.Temporary.RadioButton btnStart

Кнопка btnStop – викликає екстренну зупинку роботи системи. Для імітаційної моделі задамо момент натискання даної кнопки через 800 секунд після запуску симуляції. В умови скидування задаємо btnStart.on та btnBreakWork.on, тобто кнопка btnStop вимикається відразу після натискання однієї з цих кнопок.

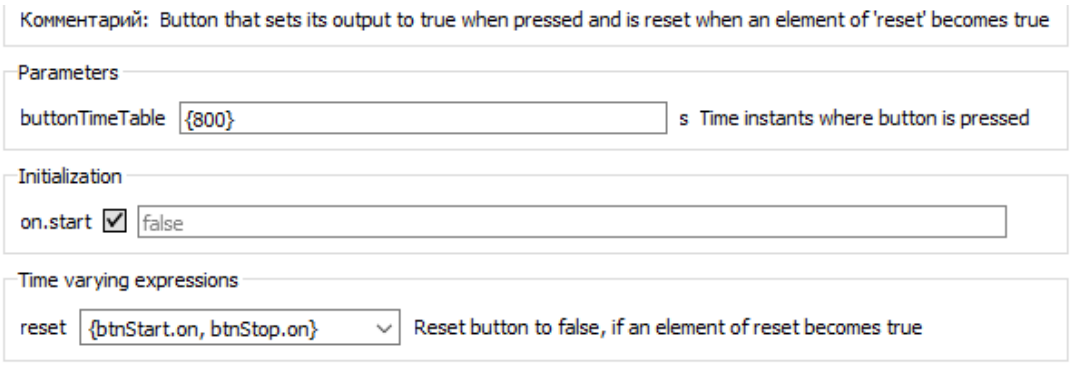

Рисунок 4.40 – Налаштування параметрів компонента Modelica.StateGraph.Temporary.RadioButton btnStop

Кнопка btnBreakWork – при натисканні негайно зупиняє робочий цикл системи та в екстренному порядку зливає рідину за баку для зберігання очищеної рідини. Для імітаційної моделі задамо момент натискання даної кнопки через 800 секунд після запуску симуляції. В умови скидування задаємо btnStart.on та btnStop.on, тобто кнопка btnBreakWork вимикається відразу після натискання однієї з цих кнопок.

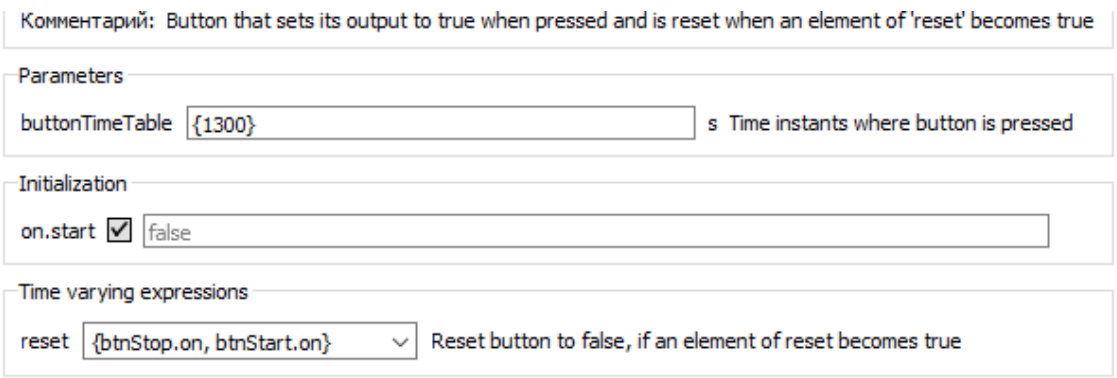

Рисунок 4.41 – Налаштування параметрів компонента Modelica.StateGraph.Temporary.RadioButton btnBreakWork

Після додавання всіх компонентів в модель, та з'єднання всіх портів – гідродинамічна АСК готова. Надалі залишилося змоделювати контролер для даної системи.

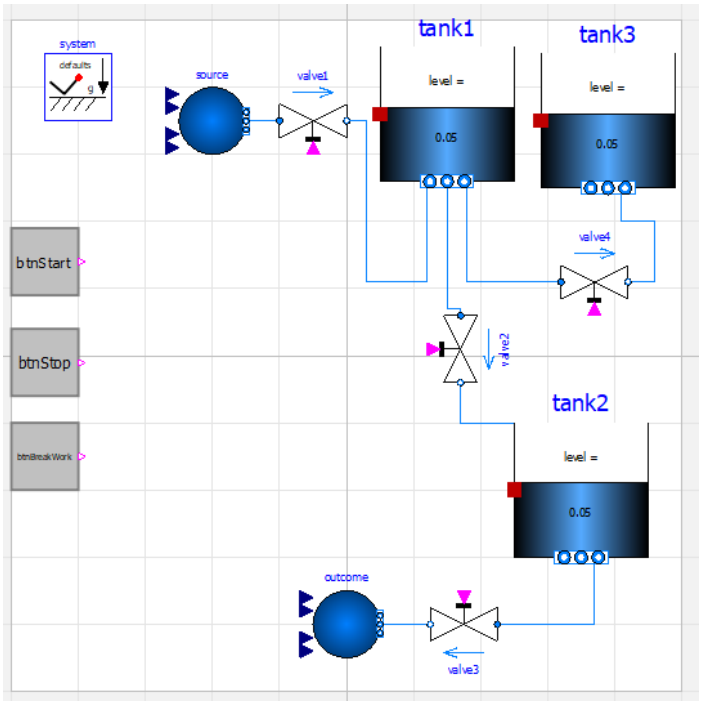

Рисунок 4.42 – Загальний вигляд побудованої моделі АСК з гідродинамічним процесом

4.4.1 Побудова моделі контролера для заданої АСК з гідродинамічним процесом

Сформулюємо вимоги до моделі контролера. Контролер повинен бути обладнаним чотирма булевими виходами для управління клапанами. Булевий вихід повертає «true» в момент часу, коли необхідно відкрити клапан. За замовчуванням значення кожного з виходів встановлене – «false». Контролер повинен бути обладнаним трьома булевими входами, по одному входу для кожної кнопки. Значення входу набуває «true» в момент часу, коли нажата кнопка. За замовчуванням всі булеві входи встановлені в положення «false». Також контролер слід обладнати двома цифровими входами. Кожен з цих входів буде приймати поточне значення рівня рідини в резервуарах «tank1» та

«tank2». Також контролер повинен мати пожливість ручного налаштування чотирьох параметрів. Перший параметр – «maxTank1_lvl», це максимальне значення наповнення першого резервуару, для того, щоб він не переповнювався. Другий параметр – «maxTank2_lvl», це максимальне значення наповнення другого резервуару, для того, щоб він не переповнювався. Третій параметр – «minTank1_lvl», це очікуване значення рівня осаду в першому резервуарі. Завдаки цьому значенню контролер буде розуміти, що в першом резервуарі залишився тільки осад, а отже чиста рідина в цьому резервуарі закінчилась. Нарешті, останній параметр – «fallingTime», цей параметр означає час відстоювання рідини (в секундах) для першого резервуару.

Опишемо нормальний робочий цикл системи для контролера. Для опису цього процесу будуть використані два компонента OpenModelica. Перший компонент – «Modelica.StateGraph.Step». Даний компонент описує звичайний крок програми (крок, який не є активним, коли починається моделювання). Другим компонентом буде «Modelica.StateGraph.Transition». Даний компонент – перехід стану, він відпрацьовує коли значення «Fire condition» встановлюється в «true» або коли відпрацьовує таймер.

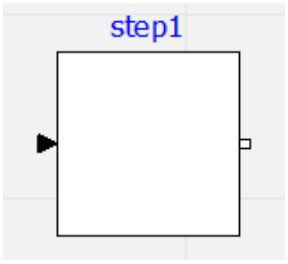

Рисунок 4.43 – Зовнішній вигляд компонента Modelica.StateGraph.Step

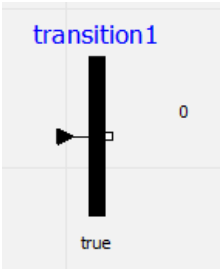

Рисунок 4.44 – Зовнішній вигляд компонента Modelica.StateGraph.Transition

Першим кроком робочого циклу алгоритму буде відкривання першого клапану, після чого рідина з джерела входу почне потрапляти в перший резервуар. Даний блок слід обладнати одним вхідним та одним вихідним портом. Вхідний порт з'єднуємо з вхідним портом «inPort».

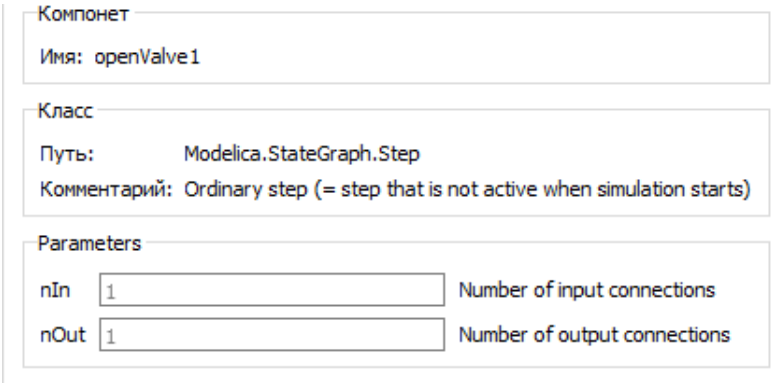

Рисунок 4.45 – Налаштування параметрів компонента Modelica.StateGraph.Step openValve1

Даний крок виконується поки перший резервуар не переповниться. Отже умова завершення даного кроку буде виглядати «Tank1_lvl > maxTank1_lvl». Для цього компонента зміни стану слід вимкнути таймер, та додати умову. Дану умову слід розуміти – завершити крок «openValve1», якщо поточний рівень рідини в першому резервуарі перевищить максимально допустимий рівень рідини для даного резервуару. Вхідний порт з'єднуємо з вихідним портом кроку «openValve1».

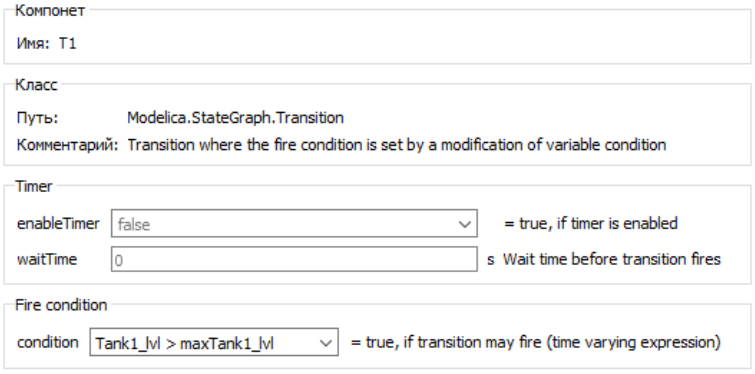

Рисунок 4.46 – Налаштування параметрів компонента Modelica.StateGraph.Transition T1

Наступним кроком робочого циклу алгоритму буде відстоювання рідини в першому резервуарі. Даний блок слід обладнати одним вхідним та одним вихідним портом. Вхідний порт з'єднуємо з вихідним портом блоку зміни стану «T1».

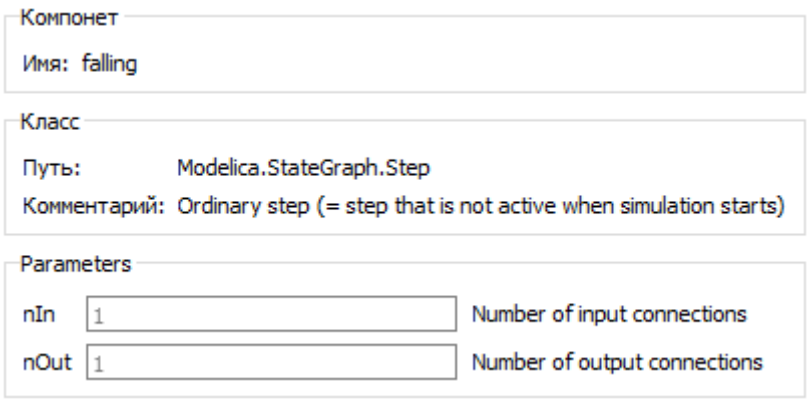

Рисунок 4.47 – Налаштування параметрів компонента Modelica.StateGraph.Step falling

Даний крок виконується поки не спрацює таймер. Дану умову слід розуміти – завершити крок «falling», якщо час очікування таймера завершився. Встановимо перемикач «enableTimer» в положення «true». А значення «waitTime» для таймера буде рівним значенню «fallingTime», яке ми задаємо для контролера. Вхідний порт з'єднуємо з вихідним портом кроку «falling».

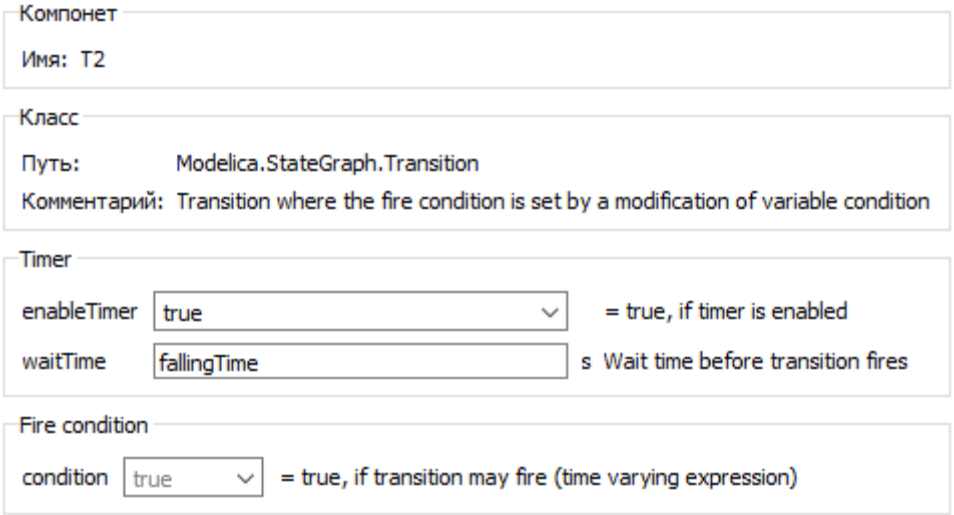

Рисунок 4.48 – Налаштування параметрів компонента Modelica.StateGraph.Transition T2

Третім кроком робочого циклу алгоритму буде відкривання другого клапану, після чого відстояна рідина з першого резервуару почне перетікати в другий резервуар. Даний блок слід обладнати одним вхідним та одним вихідним портом. Вхідний порт з'єднуємо з вихідним портом блоку зміни стану «T2».

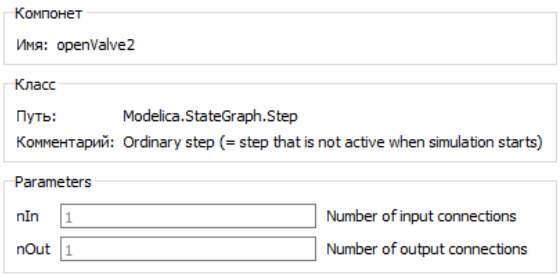

Рисунок 4.49 – Налаштування параметрів компонента Modelica.StateGraph.Step openValve2

Даний крок виконується поки в першому резервуарі не залишиться тільки осад, або другий резервуар не переповниться. Отже умова завершення даного кроку буде виглядати «Tank1_lvl < minTank1_lvl or Tank2_lvl > maxTank2_lvl». Для цього компонента зміни стану слід вимкнути таймер, та додати умову. Дану умову слід розуміти – завершити крок «openValve2», якщо поточний рівень рідини в першому резервуарі досягне мінімально допустимий рівень рідини для даного резервуару (залишиться тільки осад), або якщо поточний рівень рідини в другому резервуарі перевищить максимально допустимий рівень рідини для даного резервуару. Вхідний порт з'єднуємо з вихідним портом кроку «openValve2».

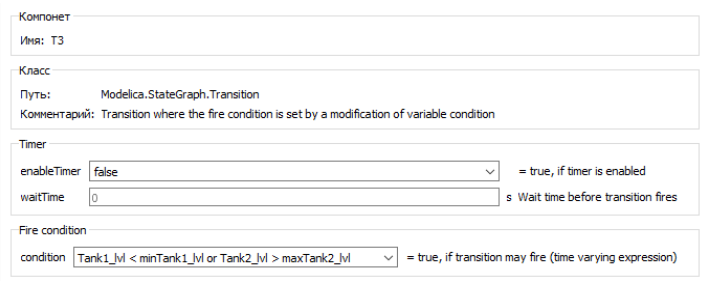

Рисунок 4.50 – Налаштування параметрів компонента Modelica.StateGraph.Transition T3

Четвертим кроком робочого циклу алгоритму буде відкривання третього клапану, після чого очищена рідина з другого резервуару почне перетікати в вихідне джерело. Даний блок слід обладнати одним вхідним та одним вихідним портом. Вхідний порт з'єднуємо з вихідним портом блоку зміни стану «T3».

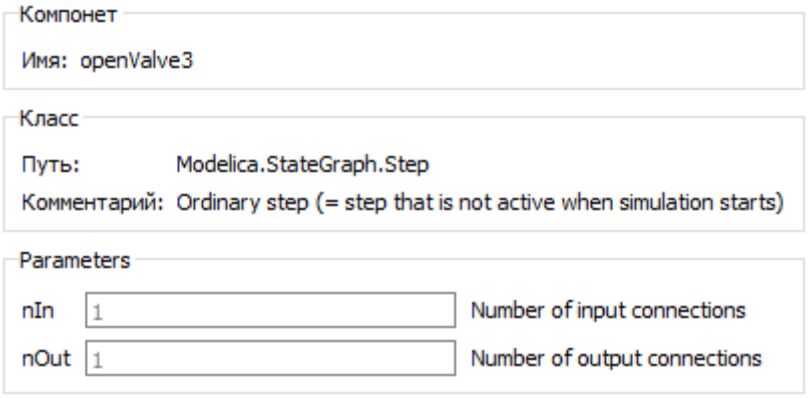

Рисунок 4.51 – Налаштування параметрів компонента Modelica.StateGraph.Step openValve3

Даний крок виконується поки в другому резервуарі залишається рідина. Отже умова завершення даного кроку буде виглядати «Tank2_lvl < 0.01». Для цього компонента зміни стану слід вимкнути таймер, та додати умову. Дану умову слід розуміти – завершити крок «openValve3», якщо поточний рівень рідини в другому резервуарі досягне мінімально допустимий рівень рідини для даного резервуару. Вхідний порт з'єднуємо з вихідним портом кроку «openValve3».

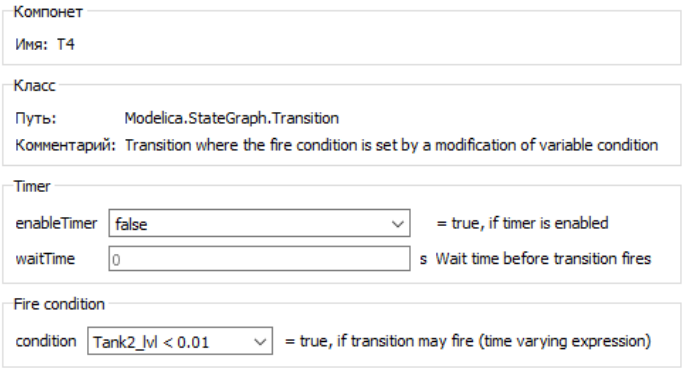

Рисунок 4.52 – Налаштування параметрів компонента Modelica.StateGraph.Transition T4

П’ятим кроком робочого циклу алгоритму буде відкривання четвертого клапану, після чого осад з першого резервуару почне перетікати в тертій резервуар. Даний блок слід обладнати одним вхідним та одним вихідним портом. Вхідний порт з’єднуємо з вихідним портом блоку зміни стану «T4».

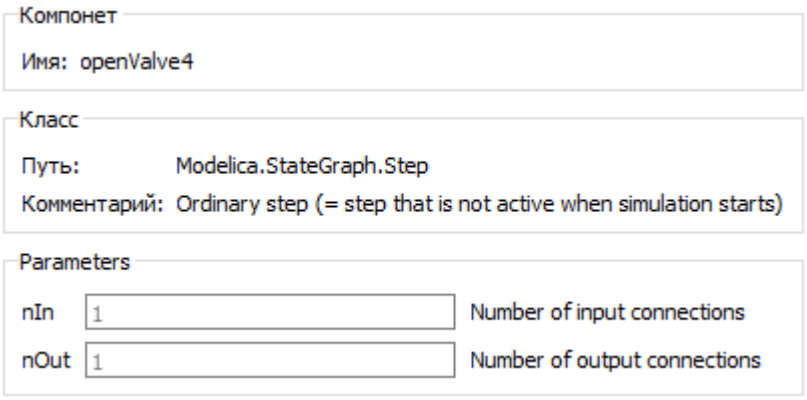

Рисунок 4.53 – Налаштування параметрів компонента Modelica.StateGraph.Step openValve4

Даний крок виконується поки в першому резервуарі залишається осад. Отже умова завершення даного кроку буде виглядати «Tank1_lvl < 0.01». Для цього компонента зміни стану слід вимкнути таймер, та додати умову. Дану умову слід розуміти – завершити крок «openValve4», якщо поточний рівень осаду в першому резервуарі досягне мінімально допустимий рівень рідини для даного резервуару. Вхідний порт з’єднуємо з вихідним портом кроку «openValve4».

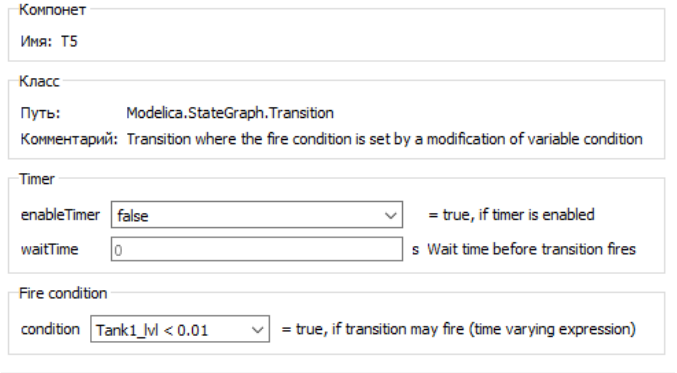

Рисунок 4.54 – Налаштування параметрів компонента Modelica.StateGraph.Transition T5

В моделі також є кнопка «btnStop» для екстренного переривання робочого циклу та зупинки роботи системи. Тому для реалізації переривання слід використати компонент – Modelica.StateGraph.PartialCompositeStep. PartialCompositeStep – стандартний інтерфейс компонента композитного кроку при моделюванні. Даний компонент обладнаний одним портом входу та одним портом виходу. Також є порт «suspend», який призначений для переривання роботи композитного кроку и «resume», який призначений для продовження роботи композитного кроку з моменту зупинки.

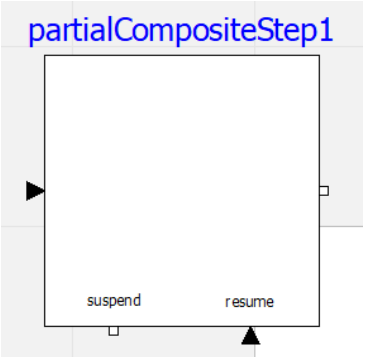

Рисунок 4.55 – Зовнішній вигляд компонента Modelica.StateGraph.PartialCompositeStep

Робочий цикл програми залежить від двох вхідних величин: «Tank1_lvl» - поточний рівень рідини в першому резервуарі, «Tank2_lvl» - поточний рівень рідини в другому резервуарі. Отже слід додати два вхідних цифрових порти до даного композитного кроку.

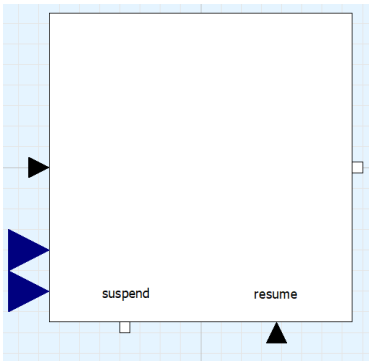

Рисунок 4.56 – Зовнішній вигляд компонента workCycleOfControllerForTypicalScheme

Інтерфейс Modelica.StateGraph.Transition не дозволяє з'єднувати цей компонент з вихідним портом композитного кроку, тому слід додати останній проміжний крок. Останнім кроком роботи робочого циклу буде «endOfStep». Даний блок слід обладнати одним вхідним та одним вихідним портом. Вхідний порт з'єднуємо з «T5», а вихідний з вихідним портом композитного кроку «outPort».

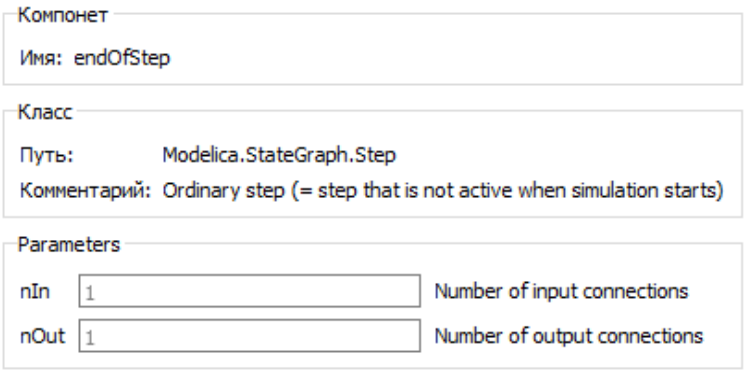

Рисунок 4.57 – Налаштування параметрів компонента Modelica.StateGraph.Step endOfStep

Загальний вигляд побудованої моделі робочого циклу системи представлено в «Модель робочого циклу контролера для АСК ДЗ». Робочий цикл роботи системи побудовано, далі слід перевірити дієздатнійсть даної моделі засобами OpenModelica. Для цього на панелі «Симуляції» слід натиснути «Перевірити модель».

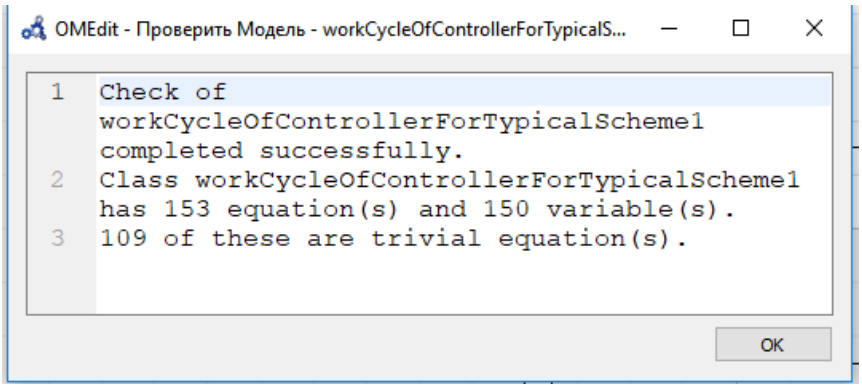

Рисунок 4.58 – Результат перевірки моделі workCycleOfControllerForTypicalScheme

Модель робочого циклу побудована та перевірена. Надалі можна розпочати моделювати контролера для даної системи використовуючи композитний крок.

Побудову моделі контролера слід розпочати з першого обов’язкового компонента - Modelica.StateGraph.StateGraphRoot. Об'єкт «StateGraphRoot» необхідний, оскільки всі об'єкти «Step» мають зовнішні посилання для зв'язку з найближчим «CompositeStep» (який успадковується від «PartialCompositeStep»), особливо для припинення «CompositeStep» через порт «suspend». Навіть якщо ніде не використано «CompositeStep», на найвищому рівні необхідне відповідне внутрішнє визначення і ініціалізація об'єкту «StateGraphRoot».

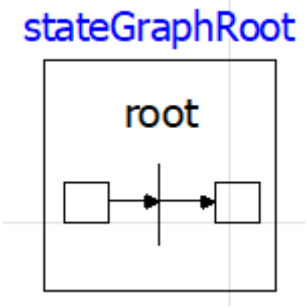

Рисунок 4.59 – Зовнішній вигляд компонента Modelica.StateGraph.StateGraphRoot

В модель контролера слід додати три булевих вхідних порти для кожної з кнопок: «btnStart», «btnStop», «btnBreak».

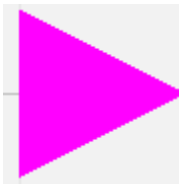

Рисунок 4.60 – Зовнішній вигляд компонента Modelica.Blocks.Interfaces.BooleanInput

Також в модель додамо два цифрових вхідних порта, завдяки яким контролер отримує інформацію про поточний рівень рідини в кожному з резервуарів: «Tank1_lvl», «Tank2_lvl».

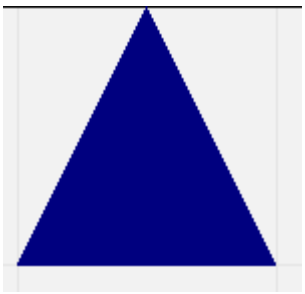

Рисунок 4.61 – Зовнішній вигляд компонента Modelica.Blocks.Interfaces.RealInput

Надалі буде описано інтерфейс для взаємодії користувача з контролером, для введення параметрів.

```
import SI = Modelica.SIunits;
parameter SI.Height maxTank1_lvl "Введіть значення максимального рівня наповнення для Tank1";
parameter SI.Height maxTank2_lvl "Введіть значення максимального рівня наповнення для Tank2";
parameter SI.Height minTank1_lvl "Введіть значення рівня осаду для Tank1";
parameter SI.Time fallingTime "Введіть час відстоювання рідини для Tank1 (секунд)";
```

Рисунок 4.62 – Оголошення обов’язкових параметрів для контролера

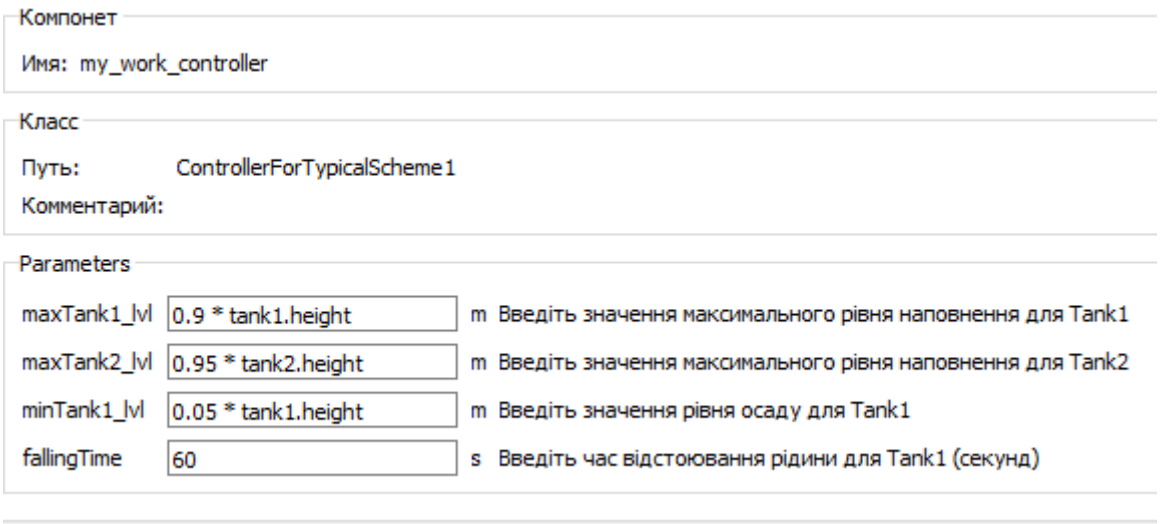

Рисунок 4.63 – Приклад ініціалізації обов’язкових параметрів для контролера

Також слід додати чотири булевих вихідних порти для управління кожного з клапанів: «out_valve1», «out_valve2», «out_valve3», «out_valve4».

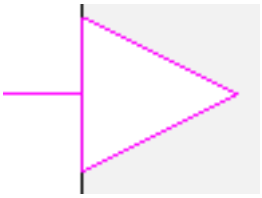

Рисунок 4.64 – Зовнішній вигляд компонента Modelica.Blocks.Interfaces.BooleanOutput

Керування значенням вихідних булевих портів відбувається за допомогою Modelica.Blocks.Sources.BooleanExpression: «open_valve1», «open_valve2», «open_valve3», «open_valve4». На перший вихідний порт подається «true», якщо виконується умова «workCycle.openValve1.active». Тобто, якщо в нормальному робочому циклі системи в цей момент часу активний крок з назвою «openValve1».

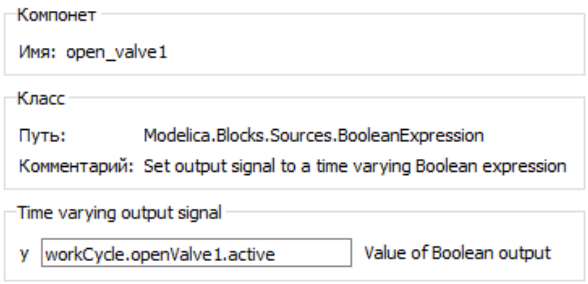

Рисунок 4.65 – Налаштування параметрів компонента Modelica.Blocks.Sources.BooleanExpression open_valve1

На другий вихідний порт подається «true», якщо виконується умова «workCycle.openValve2.active». Тобто, якщо в нормальному робочому циклі системи в цей момент часу активний крок з назвою «openValve2».

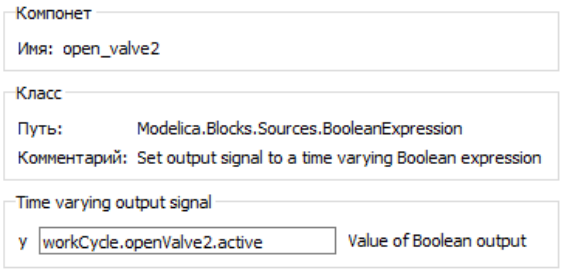

Рисунок 4.66 – Налаштування параметрів компонента Modelica.Blocks.Sources.BooleanExpression open_valve2

На третій вихідний порт подається «true», якщо виконується умова «workCycle.openValve3.active or breakWorkCycle.active». Тобто, якщо в нормальному робочому циклі системи в цей момент часу активний крок з назвою «openValve3» або крок з назвою «breakWorkCycle».

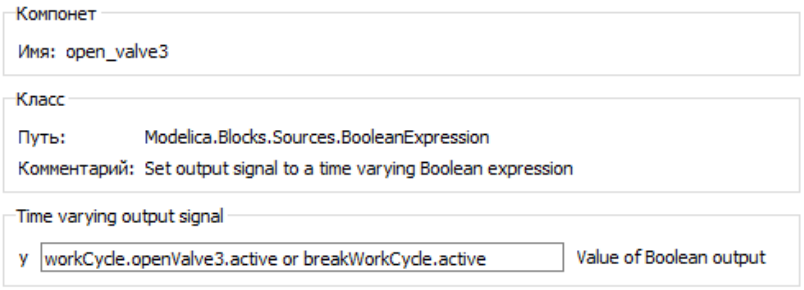

Рисунок 4.67 – Налаштування параметрів компонента Modelica.Blocks.Sources.BooleanExpression open_valve3

На останній вихідний порт подається «true», якщо виконується умова «workCycle.openValve4.active». Тобто, якщо в нормальному робочому циклі системи в цей момент часу активний крок з назвою «openValve3».

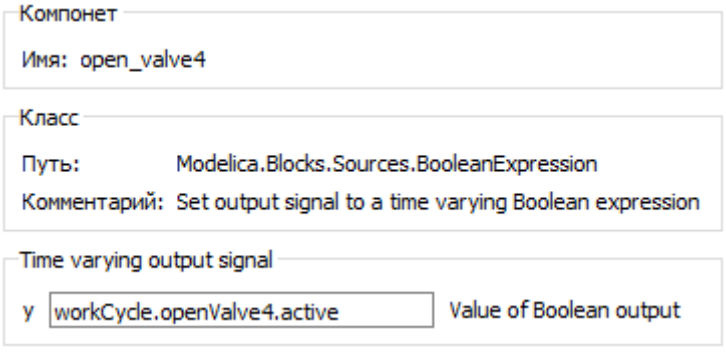

Рисунок 4.68 – Налаштування параметрів компонента Modelica.Blocks.Sources.BooleanExpression open_valve4

Наступним кроком слід підключити компоненти зміни стану до композитного кроку в контролері. Таким чином, якщо на булевий вхід контролера «btnStop» буте подано значення «true», робочий цикл зупиниться і

система буде в стані очікування наступної операції. Якщо оператор після цього натисне на кнопку «btnStart», на булевий вхід «btnStart» подасться «true» і для сиситеми знову почне виконуватися робочий цикл.

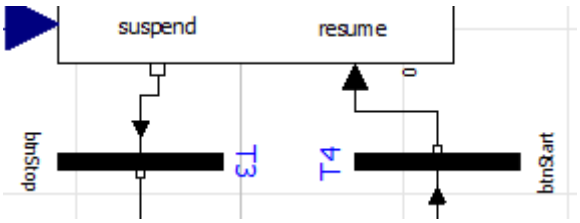

Рисунок 4.69 – Налаштування «suspend» та «resume» для робочого циклу системи

Якщо на булевий вхід контролера «btnStop» буте подано значення «true», робочий цикл зупиниться і система буде в стані очікування наступної операції. Якщо оператор після цього натисне на кнопку «btnBreakWork», на булевий вхід «btnBreak» подасться «true». Після чого виконається крок «breakWorkCycle», умою зміни стану для якого буде «Tank2_lvl < 0.01», тобто повне спустошення другого резервуару. Після чого система знову перейде в стан очікування.

Якщо оператор після цього натисне на кнопку «btnStart», на булевий вхід «btnStart» подасться «true» і для сиситеми знову почне виконуватися робочий цикл.

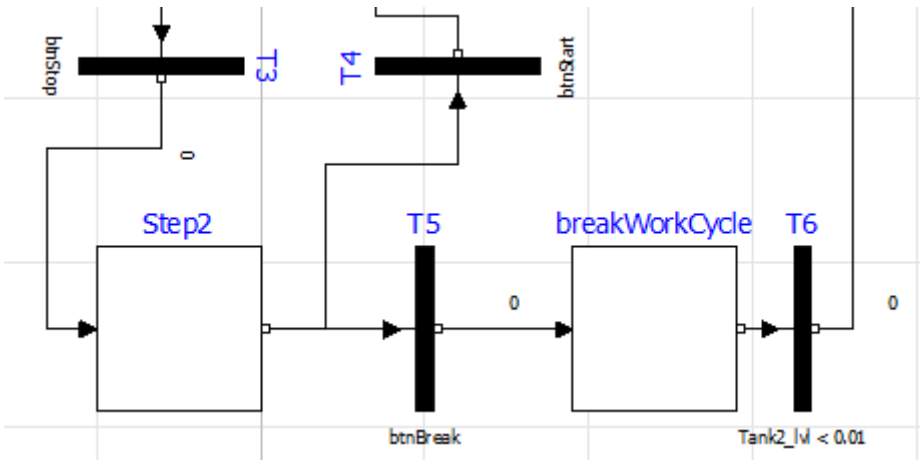

Рисунок 4.70 – Додавання в модель можливості екстренного зливу рідини з баку

Далі слід додати змодельований контролер в модель АСК та з’єднати всі порти. Наступний крок – ініціалізувати всі параметри необхідні для роботи контролера. Для цього слід ввести значення максимального рівня наповнення для «tank1» рівний 90% від максимальної висоти даного резервуара. Далі ввести значення максимального рівня наповнення для «tank2» рівний 95% від максимальної висоти цього резервуара. Потім ввести значення рівня осаду для «tank1» рівний 5% від максимальної висоти цього резервуара. Також ввести час відстоювання рідини для «tank1» рівний 60 секунд.

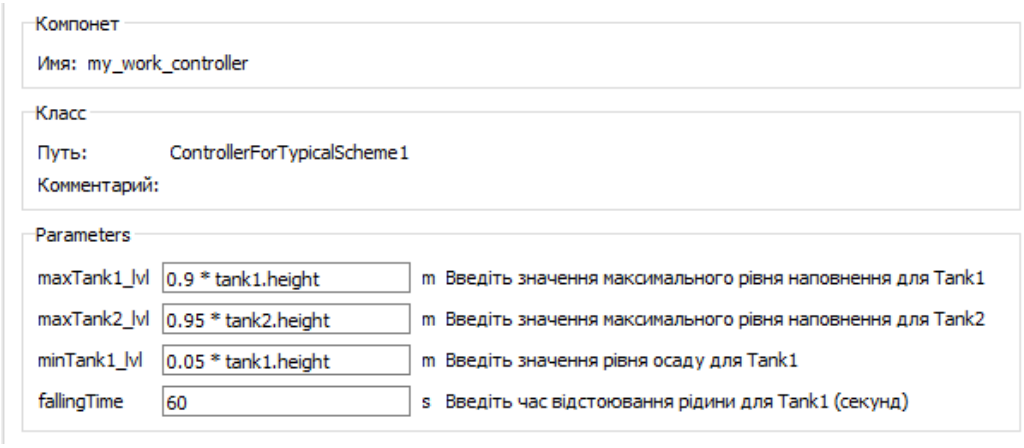

Рисунок 4.71 – Ініціалізація параметрів контролера

Готова модель представлена в додатку «Модель АСК Д1» .

4.4.2 Імітаційне моделювання запропонованої АСК

Спочатку буде перевірена коректність роботи робочого циклу системи. Для цього встановимо час натискання кнопки «btnStart» через 20 секунд після початку моделювання.

Рисунок 4.72 – Налаштування параметрів компонента btnStart

Після цього слід перевірити всі компоненти моделі. Для цього натиснути «Симуляція» потім «Перевірити всі моделі»

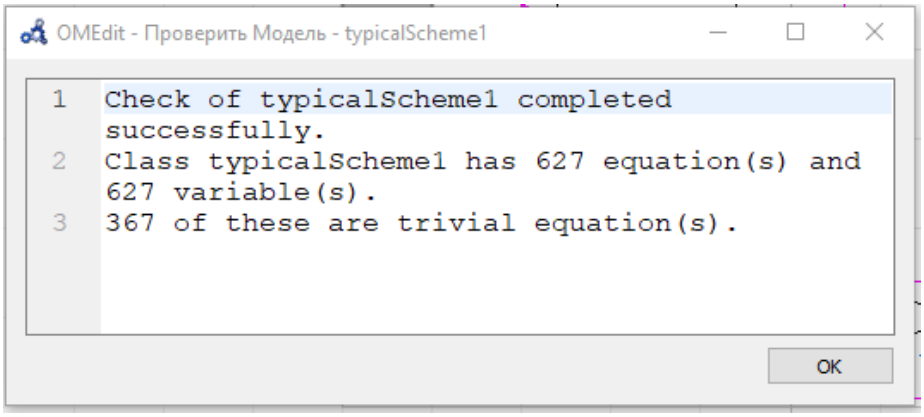

Рисунок 4.73 – Результат перевірки всіх компонентів моделі

Після цього проведемо симуляцію роботи, натиснути «Симуляція» потім «Налаштування симуляції». Та встановити час симуляції від 0 секунд до 1000 секунд, а число інтервалів симуляції – 500.

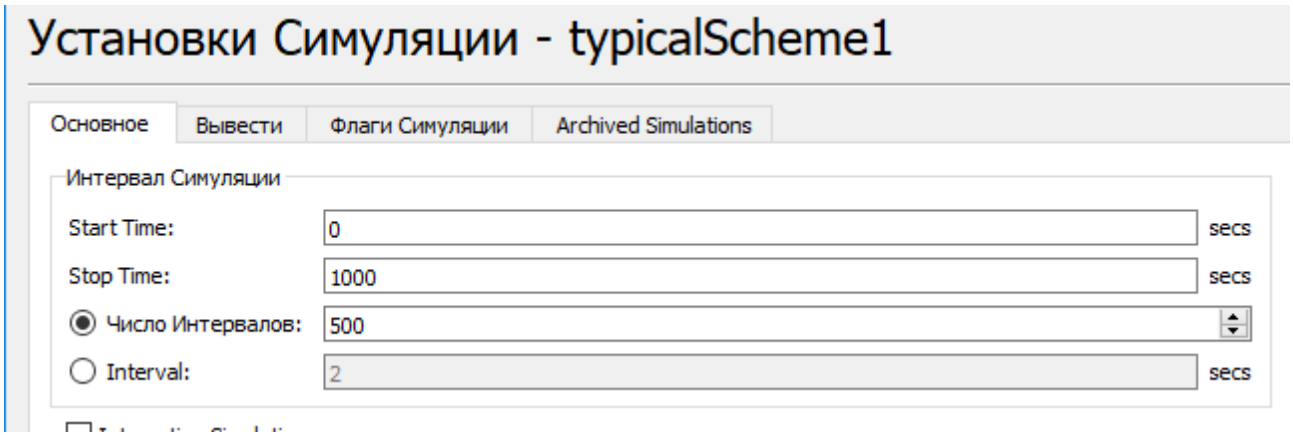

Рисунок 4.74 – Налаштування симуляції

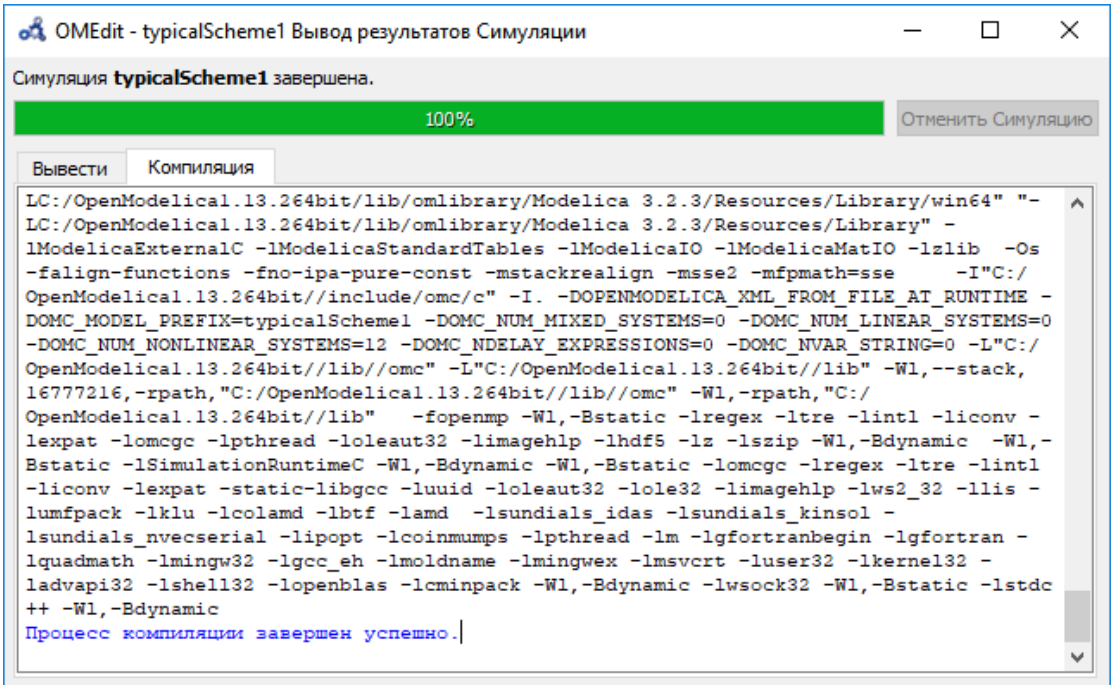

Рисунок 4.75 – Результат компіляції моделі

Далі в IDE OMEdit перейти на вкладку «Вивід на графік», та в вікні «Браузер змінних» встановити чекбокс поряд з необхідними змінними.

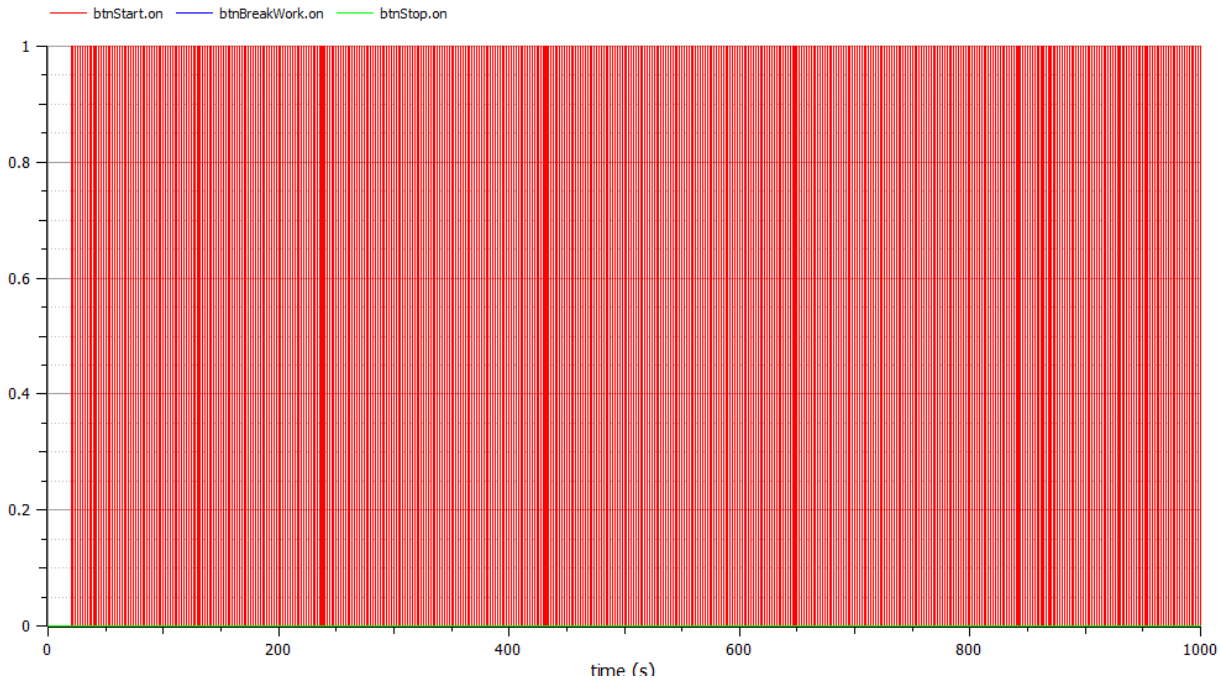

Рисунок 4.76 – Графік залежності натиснутих кнопок від часу для першого моделювання

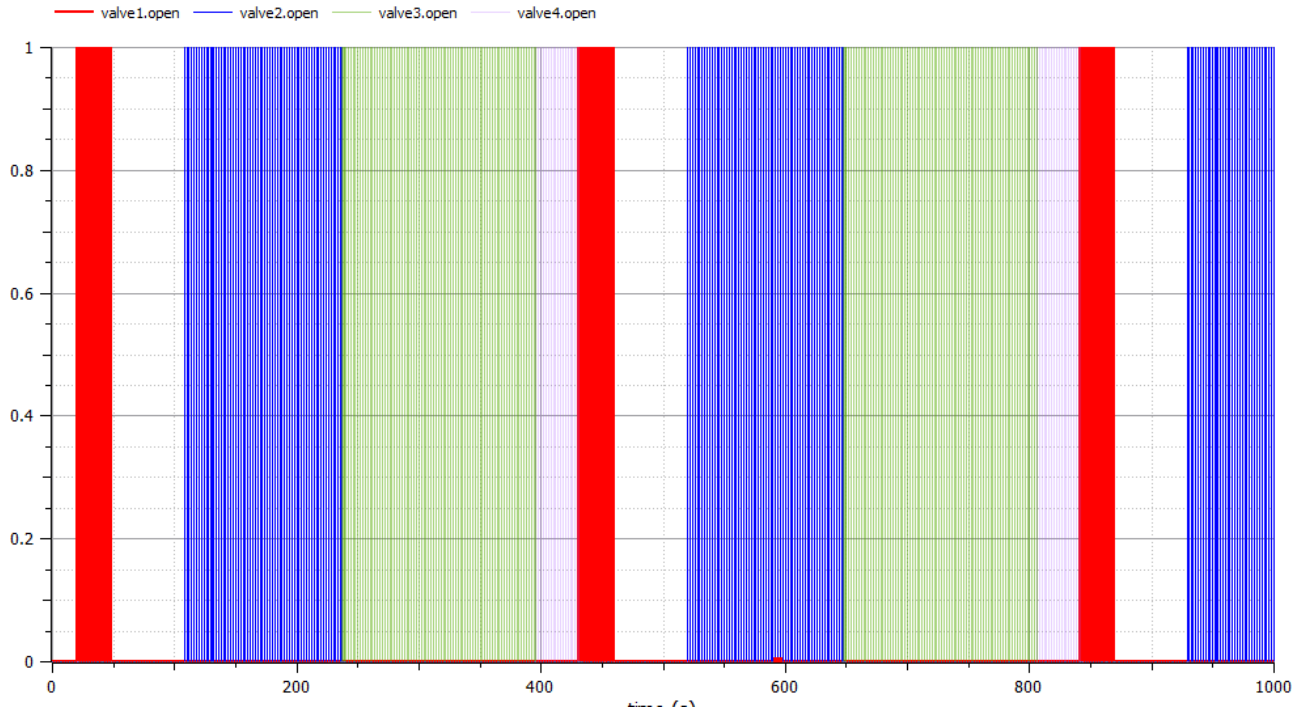

Рисунок 4.77 – Графік залежності відкриття клапанів від часу для першого моделювання

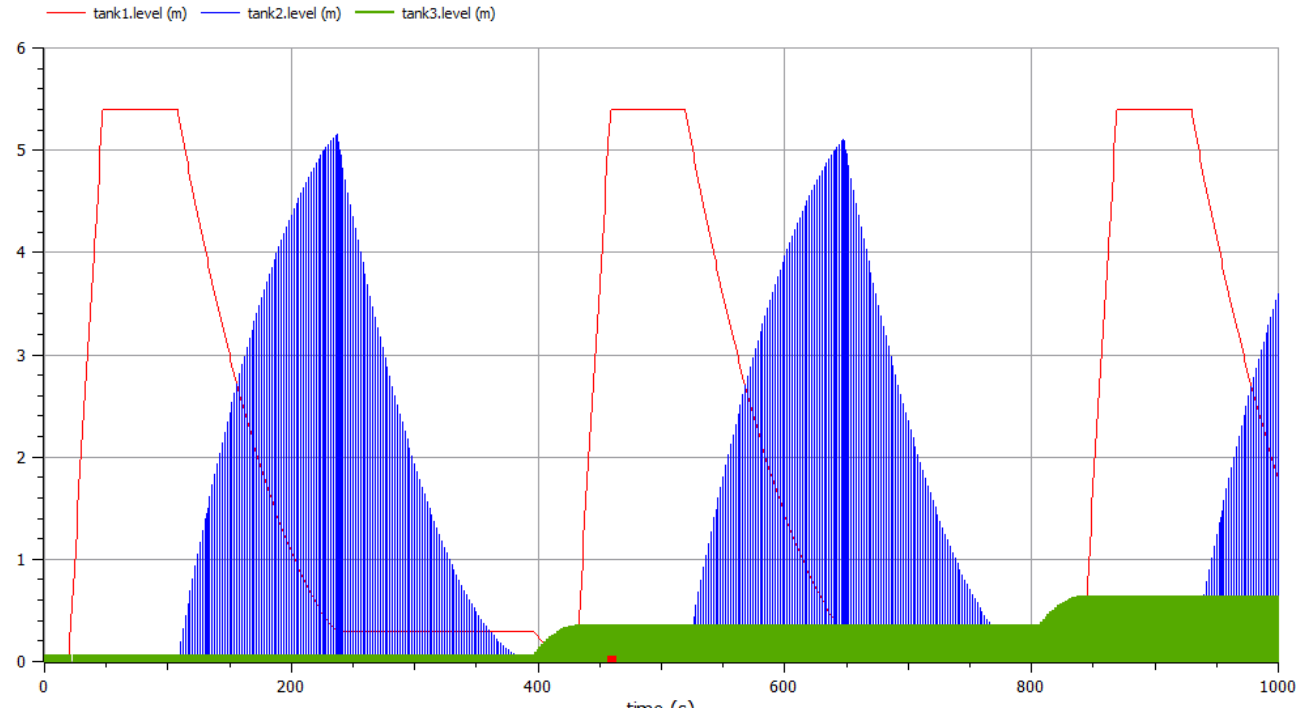

Рисунок 4.78 – Графік залежності рівня рідини в резервуарі від часу для першого моделювання

З графіків видно, що в початковий момент часу жодна з кнопок не нажата, всі клапани закриті. Після 20 секунди натискається кнопка «btnStart» і відкривається перший клапан, перший резервуар наповнюється. Через 47.7 секунд після початку симуляції рівень рідини в першому резервуарі досягає максимального значення, перший клапан. Далі запускається таймер. Через 60 секунд після запуску, перший клапан закривається і відкривається другий клапан. На 236 секунді рівень рідини в першому резервуарі досягає мінімального значення і другий клапан закривається. Далі відкривається третій клапан і очищена рідина перетікає в другий резервуар. Потім відкривається четвертий клапан і залишки осаду з першого резервуару перетікають в третій резервуар. На 431 секунді робочий цикл розпочинає роботу роботу спочатку.

Для наступного моделювання встановимо такі періоди натискання кнопок: кнопка «btnStart» натискається на 20 секунді, на 570 секунді натискається кнопка «btnStop» і система переходить в стан очікування, на 670 секунді натискається кнопка «btnBreakWork» і рідина витікає з другого резервуару в вихідне джерело. Потім на 800 секунді знову натискається кнопка «btnStart» і система знову нормально функціонує.

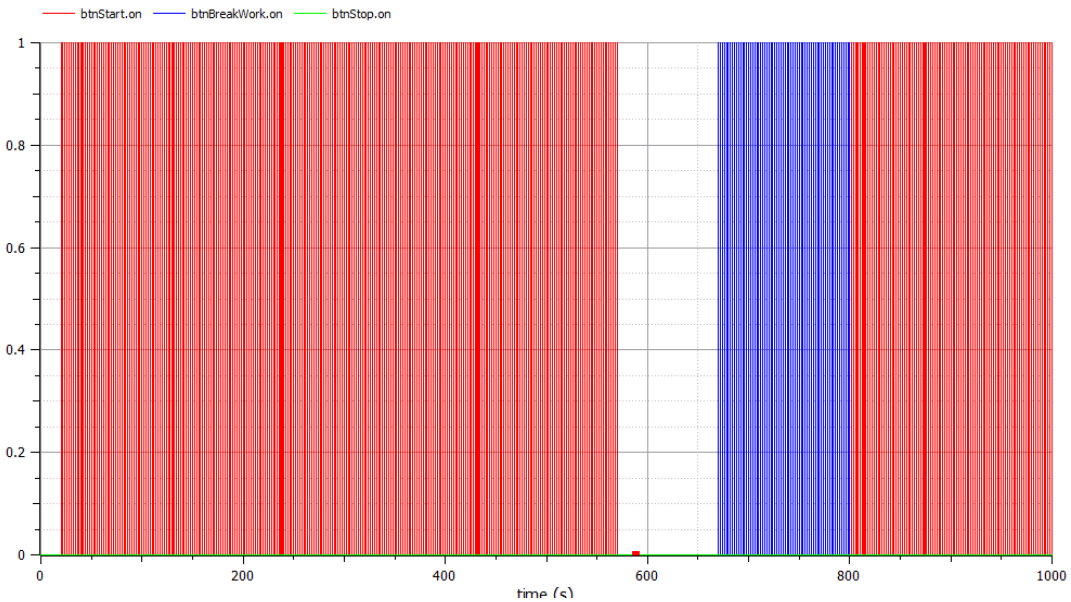

Рисунок 4.79 – Графік залежності натиснутих кнопок від часу для другого моделювання

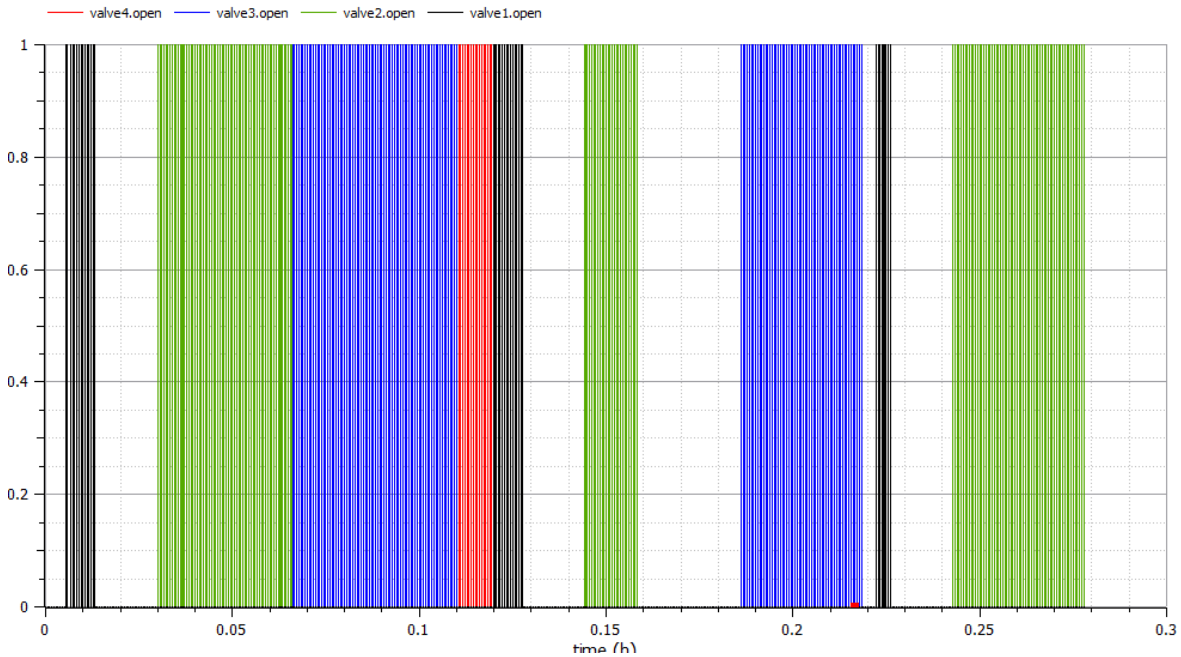

Рисунок 4.80 – Графік залежності відкриття клапанів від часу для другого моделювання

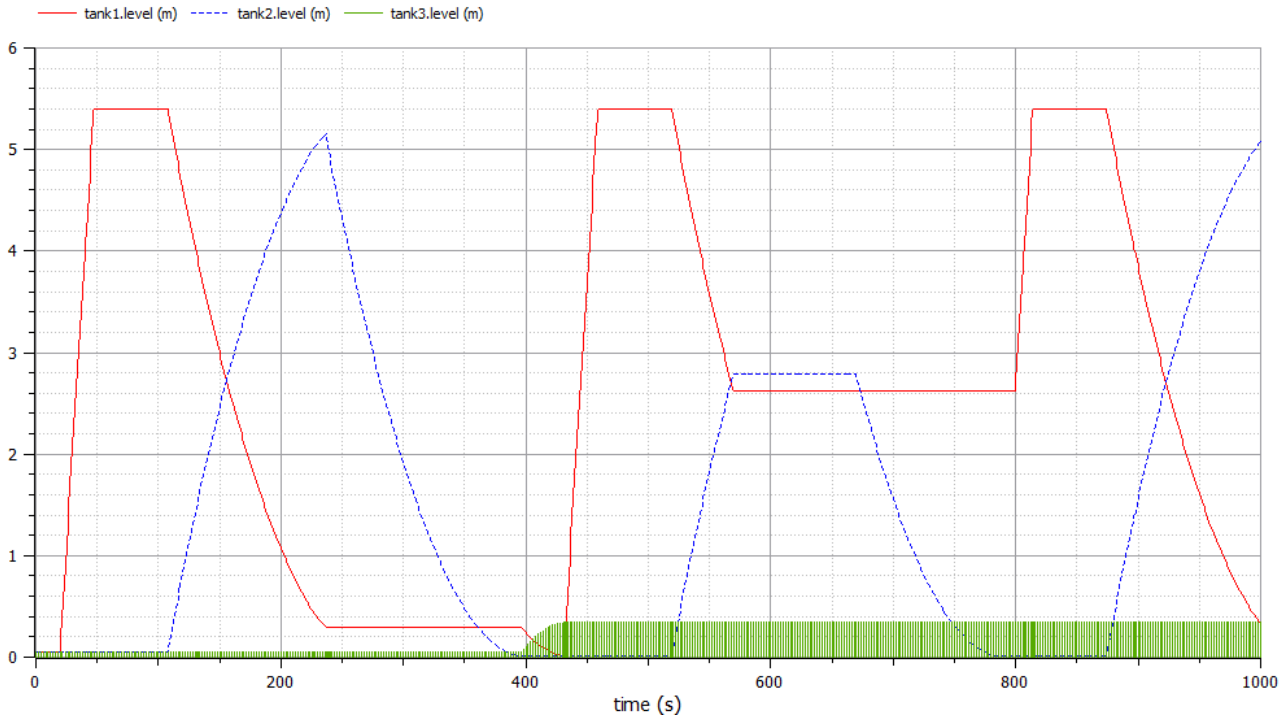

Рисунок 4.81 – Графік залежності рівня рідини в резервуарі від часу для другого моделювання

З графіків видно, що в початковий момент часу жодна з кнопок не нажата, всі клапани закриті. Після 20 секунди натискається кнопка «btnStart» і відкривається перший клапан, перший резервуар наповнюється. Через 47.7 секунд після початку симуляції рівень рідини в першому резервуарі досягає максимального значення, перший клапан. Далі запускається таймер. Через 60 секунд після запуску, перший клапан закривається і відкривається другий клапан. На 236 секунді рівень рідини в першому резервуарі досягає мінімального значення і другий клапан закривається. Далі відкривається третій клапан і очищена рідина перетікає в другий резервуар. Потім відкривається четвертий клапан і залишки осаду з першого резервуару перетікають в третій резервуар. На 431 секунді робочий цикл розпочинає роботу роботу спочатку. На 570 секунді натискається кнопка «btnStop» і система переходить в період очікування. На 670 секунді натискається кнопка «btnBreakWork» і рідина витікає з другого резервуару в вихідне джерело. Потім на 800 секунді знову натискається кнопка «btnStart» і система знову нормально функціонує.

Висновки до розділу 4

В цьому було побудовано модель електричної схеми з діодом. Після чого було використано модуль OpenModelica «Mechanics» для побудови анімованої моделі пружинного маятника. Також використавши модуль «Thermal» було змодельовано термотехнічну модель. Також в цьому розділі була описана та реалізована типова гідродинамічна АСК. Були описані її режими роботи, та модель поведінки. Врешті-решт було проведено дві імітації симуляції роботи гідродинамічної АСК.

ВИСНОВКИ

У даній роботі було досліджено ряд існуючих програмних засобів для моделювання. Всі вони мали можливість експортувати побудовану модель в формат XML. В першу чергу, було розглянуто мову моделювання SysML. SysML як розширена версія відкритого стандарту UML відкриває ряд нових можливостей. SysML цікавий можливістю генерувати код з структурних діаграм моделей. SysML більшою мірою предназначений для моделювання схем процесів, а в рамках даної роботи головний акцент був поставлений на імітаційне моделювання. Також увагу привернув відкритий стандарт AutomationML, з його можливістю будувати чітко описані моделі об'єктів. Та нажаль, інформації по AutomationML в відкритому доступі не дуже багато, отже від цього варіанту невдовзі теж довилося відмовитися.

Далі, при виборі засобу моделювання було одназначно вирішено використовувати об'єктно-орієнтовану мову моделювання Modelica. Серед переваг моделіки варто зазначити велику кількість задокументованих матеріалів, котрі допомагають освоїти даний інструмент. Першим інструментом серед Modelica-залежних мов був розглянутий інструмент Dymola. Та нажаль, майже одразі довелесь відмовитися, даний інструмент не підходив під вхідні вимоги, адже він комерційний, а демо версія не може продемонструвати всі переваги. Надалі було розглянуто UML плагін для IDE Eclipse – ModelicaML. Інструмент сам по собі є дуже цікавим. Для відображення UI частини він використовує Eclipse Papyrus Modeling environment. Сам механізм формування коду с графічних блоків-моделі реалізований на базі Acceleo. В ході дослідження ModelicaML його не вдалося навіть інсталювати, адже в залежностях для нього вказаний Acceleo версії 2.8, а на момент 2019 року, найстаріша доступна версія Acceleo – 3.0. Таким чином, можна констатувати, що на момент 2019 року ModelicaML є застарілим, недієздатним інструментом. Далі було оглянуто ще три продукти на базі мови

Modelica: JModelica.org, Wolfram SystemModeler та OpenModelica. Перший з них, а саме JModelica.org цікавий загальною інтеграцією з високорівневою мовою програмування Python. Завдяки цьому на базі одного інструменту в одній моделі можна одночасно використовувати всі переваги мов Modelica та Python. Варіант з Wolfram SystemModeler був відхилений, адже даний продукт платний. OpenModelica – реалізація з відкритим кодом для мови Modelica. Головною перевагою OpenModelica можна відмітити неймовірно широку документацію. Відвідавши https://build.openmodelica.org/Documentation/, вибір між OpenModelica та JModelica.org быльше не здавався складним. Отже вибір припав на OpenModelica. Серед переваг варто також відмітити, що це продукт з відкритим кодом, велику бібліотеку стандартних класів та просту можливість змінювати та створювавати нові класи-модулі. Отже, в даному проекті для моделювання АСК була використована OpenModelica з її безкоштовним IDE - «Openmodelica Connection Editor».

Далі була описана структурна схема типової гідродинамічної АСК, та всіх процесів, які в ній відбуваються.

Після вибору інструмента відбулося знайомство з ним. Для знайомства з усіма можливостями була побудована модель простої електричної схеми, для відображення роботи моделі діода.

Далі відбувалося знайомство з побудовою анімацій імітаційних моделей механічних процесів стандартними засобами OpenModelica. Саме тоді була побудована анімована імітаційна модель пружинного маятника.

Наступним кроком досліджувалася модель теплотехнічного процесу, і те, наскільки OpenModelica підходить для цього.

В останньому розділі, засобами OpenModelica, була побудована складна автоматизована система керування гідродинамічним процесом, яка керувалася за допомогою змодельованого контролера. Після успішного моделювання та симулювання роботи АСК, дана модель була експортована в XML формат. Код моделі на мові Modelica представлено у ДОДАТОК А.

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

1. Автоматизированная система управления [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Автоматизированная_система_управления.

2. Боев В.Д. Компьютерное моделирование систем / Василий Боев – м. Москва, 2019.

3. Робота із XML [Електронний ресурс] – Режим доступу до ресурсу: https://helpx.adobe.com/ua/incopy/using/xml.html.

4. SysML [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/SysML.

5. Диаграммы SysML [Електронний ресурс] – Режим доступу до ресурсу: http://sewiki.ru/SysML.

6. Modeling Languages for Requirements Engineering and Quantitative Analysis of Embedded Systems [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/260433525_Modeling_Languages_for_Requirements_Engineering_and_Quantitative_Analysis_of_Embedded_Systems.

7. AutomationML [Електронний ресурс] – Режим доступу до ресурсу: https://de.wikipedia.org/wiki/AutomationML.

8. Kovalenko O. AutomationML Ontology: Modeling Cyber-Physical Systems for Industry 4.0 / Olga Kovalenko, Irlán Grangel-González, Marta Sabou – м. Vienna, 2009.

9. Modelica [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Modelica.

10. ModelicaML - A UML Profile for Modelica [Електронний ресурс] – Режим доступу до ресурсу: https://openmodelica.org/?id=139:modelicaml.

11. Pop A. Eclipse Support for Design and Requirements Engineering Based on ModelicaML / Adrian Pop, Vasile Băluţă, Peter Fritzson – м. Linköping.

12. Schamai W. Modelica Modeling Language (ModelicaML) / Wladimir Schamai – м. Linköping, 2009.

13. JModelica.org [Електронний ресурс] – Режим доступу до ресурсу: https://jmodelica.org/.

14. JModelica.org [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/JModelica.org.

15. Wolfram SystemModeler [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Wolfram_SystemModeler.

16. DYMOLA и проектирование систем [Електронний ресурс] – Режим доступу до ресурсу: https://www.3ds.com/ru/produkty-i-uslugi/catia/produkty/dymola/.

ДОДАТОК А. Лістинг побудованої на мові Modelica моделі гідродинамічної АСК

```
model typicalScheme1
"Імітаційне моделювання типової АСК1"
  extends Modelica.Icons.Example;
  package Medium = Modelica.Media.Water.ConstantPropertyLiquidWater;
  Modelica.Fluid.Valves.ValveDiscrete valve1(                redeclare
     package Medium = Medium,
    m_flow_nominal=40,
    dp_nominal=100000)
    annotation (Placement(transformation(
        origin={-10,70},
        extent={{10,-10},{-10,10}},
        rotation=180)));
  Modelica.Fluid.Vessels.OpenTank tank1(

    redeclare package Medium = Medium,
    crossArea= 5,
    height= 6,level_start=0.05,
    nPorts= 3,
    portsData={Modelica.Fluid.Vessels.BaseClasses.VesselPortsData(
          diameter=0.2,
          height=4,
          zeta_out=0,
          zeta_in=1), Modelica.Fluid.Vessels.BaseClasses.VesselPortsData(diameter =
0.2, height = 0, zeta_out = 0, zeta_in = 1),
Modelica.Fluid.Vessels.BaseClasses.VesselPortsData(diameter = 0.2, height = 0,
```

```
zeta_out = 0, zeta_in = 1)}) annotation (Placement(visible = true,
transformation(extent = {{10, 52}, {50, 92}}, rotation = 0)));
  Modelica.Blocks.Sources.RealExpression level1(y=tank1.level)
    annotation (Placement(visible = true, transformation(extent = {{-90, -60}, {-55, -
40}}, rotation = 0)));
  Modelica.Fluid.Valves.ValveDiscrete valve2(       redeclare package Medium
      = Medium,
    dp_nominal(displayUnit="Pa") = 1,
    m_flow_nominal=100)
    annotation (Placement(visible = true, transformation(origin = {34, 2}, extent =
{{10, -10}, {-10, 10}}, rotation = 90)));
  Modelica.Fluid.Valves.ValveDiscrete valve3(       redeclare package Medium
      = Medium,
    dp_nominal(displayUnit="Pa") = 1,
    m_flow_nominal=10)
    annotation (Placement(transformation(
        origin={35,-80},
        extent={{10,-10},{-10,10}})));
  Modelica.Fluid.Vessels.OpenTank tank2(

    redeclare package Medium = Medium,
    crossArea= 5,
    height= 25,level_start=0.05,
    nPorts=2,
    portsData={Modelica.Fluid.Vessels.BaseClasses.VesselPortsData(
          diameter=0.2,
          height=5,
          zeta_out=0,
          zeta_in=1),Modelica.Fluid.Vessels.BaseClasses.VesselPortsData(
```

```
        diameter=0.2,
        height=0,
        zeta_out=0,
        zeta_in=1)}) annotation (Placement(transformation(extent={{50,-60},
          {90,-20}})));
  Modelica.Fluid.Sources.Boundary_pT outcome(redeclare package Medium =
      Medium,nPorts=1,
    p=system.p_ambient,
    T=system.T_ambient)
    annotation (Placement(transformation(extent={{-10,-90},{10,-70}})));
  Modelica.Blocks.Sources.RealExpression level2(y=tank2.level)
    annotation (Placement(transformation(extent={{-70,-80},{-33,-60}})));
  Modelica.Fluid.Sources.Boundary_pT source(redeclare package Medium =
      Medium, p=2.5e6,nPorts=1,
    T=system.T_ambient)
    annotation (Placement(transformation(
      origin={-40,70},
      extent={{-10,-10},{10,10}})));
  inner Modelica.Fluid.System
system(energyDynamics=Modelica.Fluid.Types.Dynamics.FixedInitial)
                annotation (Placement(transformation(extent={{-90,70},
          {-70,90}})));
  ControllerForTypicalScheme1 my_work_controller(fallingTime = 60,maxTank1_lvl
= 0.9 * tank1.height, maxTank2_lvl = 0.95 * tank2.height, minTank1_lvl = 0.05 *
tank1.height)  annotation(
    Placement(visible = true, transformation(origin = {-32, -2}, extent = {{-20, -20},
{20, 20}}, rotation = 0)));
  Modelica.Fluid.Valves.ValveDiscrete valve4(redeclare package Medium = Medium,
dp_nominal (displayUnit = "Pa") = 1, m_flow_nominal = 100)  annotation(
```

```
    Placement(visible = true, transformation(origin = {74, 22}, extent = {{10, -10}, {-
10, 10}}, rotation = 180)));
  Modelica.Fluid.Vessels.OpenTank tank3(redeclare package Medium = Medium,
crossArea = 5, height = 6, level_start = 0.05, nPorts = 1, portsData =
{Modelica.Fluid.Vessels.BaseClasses.VesselPortsData(diameter = 0.2, height = 4,
zeta_out = 0, zeta_in = 1)}) annotation(
    Placement(visible = true, transformation(extent = {{58, 50}, {98, 90}}, rotation =
0)));
  Modelica.Fluid.Examples.ControlledTankSystem.Utilities.RadioButton
btnBreakWork(buttonTimeTable = {1300}, reset = {btnStop.on, btnStart.on})
annotation(
    Placement(visible = true, transformation(extent = {{-100, -40}, {-80, -20}},
rotation = 0)));
  Modelica.Fluid.Examples.ControlledTankSystem.Utilities.RadioButton
btnStop(buttonTimeTable = {140}, reset = {btnStart.on, btnStop.on})  annotation(
    Placement(visible = true, transformation(extent = {{-100, -12}, {-80, 8}}, rotation
= 0)));
  Modelica.Fluid.Examples.ControlledTankSystem.Utilities.RadioButton
btnStart(buttonTimeTable = {20, 200}, reset = {btnStop.on, btnBreakWork.on})
annotation(
    Placement(visible = true, transformation(extent = {{-100, 18}, {-80, 38}}, rotation
= 0)));
equation
  connect(my_work_controller.btnStart, btnStart.on) annotation(
    Line(points = {{-56, 10}, {-70, 10}, {-70, 28}, {-78, 28}, {-78, 28}}, color =
{255, 0, 255}));
  connect(my_work_controller.btnStop, btnStop.on) annotation(
    Line(points = {{-56, -2}, {-80, -2}, {-80, -2}, {-78, -2}}, color = {255, 0, 255}));
  connect(my_work_controller.btnBreak, btnBreakWork.on) annotation(
```

```
    Line(points = {{-56, -14}, {-70, -14}, {-70, -30}, {-80, -30}, {-80, -30}, {-78, -30}}, color = {255, 0, 255}));
  connect(level1.y, my_work_controller.Tank1_lvl) annotation(
    Line(points = {{-54, -50}, {-42, -50}, {-42, -26}}, color = {0, 0, 127}));
  connect(valve4.port_b, tank3.ports[1]) annotation(
    Line(points = {{84, 22}, {92, 22}, {92, 40}, {82, 40}, {82, 50}, {78, 50}}, color = {0, 127, 255}));
  connect(tank1.ports[3], valve4.port_a) annotation(
    Line(points = {{30, 52}, {36, 52}, {36, 22}, {64, 22}, {64, 22}}, color = {0, 127, 255}));
  connect(valve4.open, my_work_controller.out_valve4) annotation(
    Line(points = {{74, 14}, {74, 14}, {74, 0}, {94, 0}, {94, -98}, {-16, -98}, {-16, -52}, {0, -52}, {0, -18}, {-10, -18}, {-10, -18}}, color = {255, 0, 255}));
  connect(valve2.port_a, tank1.ports[2]) annotation(
    Line(points = {{34, 12}, {34, 14}, {30, 14}, {30, 52}}, color = {0, 127, 255}));
  connect(valve1.port_b, tank1.ports[1]) annotation(
    Line(points = {{0, 70}, {6, 70}, {6, 22}, {24, 22}, {24, 52}, {30, 52}}, color = {0, 127, 255}));
  connect(valve1.open, my_work_controller.out_valve1) annotation(
    Line(points = {{-10, 62}, {-10, 62}, {-10, 38}, {2, 38}, {2, 14}, {-10, 14}, {-10, 14}}, color = {255, 0, 255}));
  connect(valve2.open, my_work_controller.out_valve2) annotation(
    Line(points = {{26, 2}, {-10, 2}}, color = {255, 0, 255}));
  connect(valve2.port_b, tank2.ports[1]) annotation(
    Line(points = {{34, -8}, {34, -20}, {50, -20}, {50, -60}, {66, -60}}, color = {0, 127, 255}));
  connect(valve3.open, my_work_controller.out_valve3) annotation(
    Line(points = {{36, -72}, {12, -72}, {12, -8}, {-10, -8}, {-10, -8}}, color = {255, 0, 255}));
```

```
  connect(level2.y, my_work_controller.Tank2_lvl) annotation(
    Line(points = {{-32, -70}, {-20, -70}, {-20, -26}, {-22, -26}}, color = {0, 0,
127}));
  connect(source.ports[1], valve1.port_a) annotation(
    Line(points = {{-30, 70}, {-20, 70}}, color = {0, 127, 255}));
  connect(valve3.port_b, outcome.ports[1]) annotation(
    Line(points = {{25, -80}, {10, -80}}, color = {0, 127, 255}));
  connect(tank2.ports[2], valve3.port_a) annotation(
    Line(points = {{74, -60}, {74, -80}, {45, -80}}, color = {0, 127, 255}));
  annotation(
    experiment(StopTime = 900),
    uses(Modelica(version = "3.2.3")));
end typicalScheme1;


model ControllerForTypicalScheme1
  extends Modelica.StateGraph.Interfaces.PartialStateGraphIcon;
  import SI = Modelica.SIunits;
  parameter SI.Height maxTank1_lvl "Введіть значення максимального рівня
наповнення для Tank1";
  parameter SI.Height maxTank2_lvl "Введіть значення максимального рівня
наповнення для Tank2";
  parameter SI.Height minTank1_lvl "Введіть значення рівня осаду для Tank1";
  parameter SI.Time fallingTime "Введіть час відстоювання рідини для Tank1
(секунд)";

  Modelica.Blocks.Interfaces.BooleanOutput out_valve1 annotation(
```

```
    Placement(visible = true, transformation(origin = {110, 76}, extent = {{-10, -10},
{10, 10}}, rotation = 0), iconTransformation(origin = {110, 76}, extent = {{-10, -
10}, {10, 10}}, rotation = 0)));
  Modelica.Blocks.Interfaces.BooleanOutput out_valve2 annotation(
    Placement(visible = true, transformation(origin = {110, 24}, extent = {{-10, -10},
{10, 10}}, rotation = 0), iconTransformation(origin = {110, 24}, extent = {{-10, -
10}, {10, 10}}, rotation = 0)));
  Modelica.Blocks.Interfaces.BooleanOutput out_valve3 annotation(
    Placement(visible = true, transformation(origin = {110, -30}, extent = {{-10, -10},
{10, 10}}, rotation = 0), iconTransformation(origin = {110, -30}, extent = {{-10, -
10}, {10, 10}}, rotation = 0)));
  Modelica.Blocks.Sources.BooleanExpression open_valve1(y =
workCycle.openValve1.active) annotation(
    Placement(visible = true, transformation(origin = {66, 90}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  Modelica.Blocks.Sources.BooleanExpression open_valve2(y =
workCycle.openValve2.active)  annotation(
    Placement(visible = true, transformation(origin = { 66, 76}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  Modelica.Blocks.Sources.BooleanExpression open_valve3(y =
workCycle.openValve3.active or breakWorkCycle.active)  annotation(
    Placement(visible = true, transformation(origin = {66, 62}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  Modelica.Blocks.Interfaces.RealInput Tank1_lvl annotation(
    Placement(visible = true, transformation(origin = {-50, -120}, extent = {{-20, -
20}, {20, 20}}, rotation = 90), iconTransformation(origin = {-50, -120}, extent = {{-
20, -20}, {20, 20}}, rotation = 90)));
  Modelica.Blocks.Interfaces.RealInput Tank2_lvl annotation(
```

```
    Placement(visible = true, transformation(origin = {52, -120}, extent = {{-20, -20},
{20, 20}}, rotation = 90), iconTransformation(origin = {52, -120}, extent = {{-20, -
20}, {20, 20}}, rotation = 90)));
  Modelica.StateGraph.InitialStep S1(nIn = 2)  annotation(
    Placement(visible = true, transformation(origin = {-64, 0}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  Modelica.StateGraph.Transition T1(condition = btnStart)  annotation(
    Placement(visible = true, transformation(origin = {-40, 0}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  Modelica.StateGraph.Transition T2(condition = btnStart) annotation(
    Placement(visible = true, transformation(origin = {40, 0}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  inner Modelica.StateGraph.StateGraphRoot stateGraphRoot annotation(
    Placement(visible = true, transformation(origin = {-86, 84}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  Modelica.Blocks.Interfaces.BooleanInput btnStart annotation(
    Placement(visible = true, transformation(origin = {-120, 60}, extent = {{-20, -20},
{20, 20}}, rotation = 0), iconTransformation(origin = {-120, 60}, extent = {{-20, -
20}, {20, 20}}, rotation = 0)));
  Modelica.Blocks.Interfaces.BooleanInput btnStop annotation(
    Placement(visible = true, transformation(origin = {-120, 0}, extent = {{-20, -20},
{20, 20}}, rotation = 0), iconTransformation(origin = {-120, 0}, extent = {{-20, -20},
{20, 20}}, rotation = 0)));
  Modelica.Blocks.Interfaces.BooleanInput btnBreak annotation(
    Placement(visible = true, transformation(origin = {-120, -60}, extent = {{-20, -
20}, {20, 20}}, rotation = 0), iconTransformation(origin = {-120, -60}, extent = {{-
20, -20}, {20, 20}}, rotation = 0)));
  Modelica.StateGraph.Transition T3(condition = btnStop)  annotation(
```

```
    Placement(visible = true, transformation(origin = {-12, -36}, extent = {{-10, -10},
{10, 10}}, rotation = -90)));
  Modelica.StateGraph.Transition T4(condition = btnStart)  annotation(
    Placement(visible = true, transformation(origin = {24, -36}, extent = {{-10, -10},
{10, 10}}, rotation = 90)));
  Modelica.StateGraph.Step S2(nOut = 2)  annotation(
    Placement(visible = true, transformation(origin = {-10, -68}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  Modelica.StateGraph.Step breakWorkCycle annotation(
    Placement(visible = true, transformation(origin = {48, -68}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  Modelica.StateGraph.Transition T5(condition = btnBreak)  annotation(
    Placement(visible = true, transformation(origin = {20, -68}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  Modelica.StateGraph.Transition T6(condition = Tank2_lvl < 0.01) annotation(
    Placement(visible = true, transformation(origin = {66, -68}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  Modelica.Blocks.Interfaces.BooleanOutput out_valve4 annotation(
    Placement(visible = true, transformation(origin = {112, -78}, extent = {{-10, -10},
{10, 10}}, rotation = 0), iconTransformation(origin = {112, -78}, extent = {{-10, -
10}, {10, 10}}, rotation = 0)));
  Modelica.Blocks.Sources.BooleanExpression open_valve4(y =
workCycle.openValve4.active)  annotation(
    Placement(visible = true, transformation(origin = {66, 46}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  workCycleOfControllerForTypicalScheme1 workCycle(fallingTime = fallingTime,
maxTank1_lvl = maxTank1_lvl, maxTank2_lvl = maxTank2_lvl, minTank1_lvl =
minTank1_lvl)  annotation(
```

```
    Placement(visible = true, transformation(origin = {2, -2.22045e-16}, extent = {{-16, -16}, {16, 16}}, rotation = 0)));
equation
  connect(Tank1_lvl, workCycle.Tank1_lvl) annotation(
    Line(points = {{-50, -120}, {-50, -120}, {-50, -16}, {-32, -16}, {-32, -12}, {-26, -12}, {-26, -12}}, color = {0, 0, 127}));
  connect(Tank2_lvl, workCycle.Tank2_lvl) annotation(
    Line(points = {{52, -120}, {52, -120}, {52, -84}, {-34, -84}, {-34, -20}, {-26, -20}, {-26, -20}}, color = {0, 0, 127}));
  connect(workCycle.inPort, T1.outPort) annotation(
    Line(points = {{-24, 0}, {-38, 0}, {-38, 0}, {-38, 0}}));
  connect(T3.inPort, workCycle.suspend[1]) annotation(
    Line(points = {{-12, -32}, {-12, -32}, {-12, -28}, {-10, -28}, {-10, -24}, {-10, -24}}));
  connect(T4.outPort, workCycle.resume[1]) annotation(
    Line(points = {{24, -34}, {24, -34}, {24, -30}, {14, -30}, {14, -26}, {14, -26}}));
  connect(T2.inPort, workCycle.outPort) annotation(
    Line(points = {{36, 0}, {26, 0}, {26, 0}, {26, 0}}));
  connect(open_valve2.y, out_valve2) annotation(
    Line(points = {{78, 76}, {92, 76}, {92, 24}, {110, 24}}, color = {255, 0, 255}));
  connect(open_valve1.y, out_valve1) annotation(
    Line(points = {{77, 90}, {96, 90}, {96, 76}, {110, 76}}, color = {255, 0, 255}));
  connect(open_valve3.y, out_valve3) annotation(
    Line(points = {{77, 62}, {88, 62}, {88, -30}, {110, -30}}, color = {255, 0, 255}));
  connect(open_valve4.y, out_valve4) annotation(
    Line(points = {{78, 46}, {82, 46}, {82, -78}, {104, -78}, {104, -78}, {112, -78}}, color = {255, 0, 255}));
  connect(T6.outPort, S1.inPort[2]) annotation(
```

```
    Line(points = {{68, -68}, {72, -68}, {72, 38}, {-86, 38}, {-86, 0}, {-74, 0}, {-74,
0}}));
  connect(T2.outPort, S1.inPort[1]) annotation(
    Line(points = {{42, 0}, {58, 0}, {58, 34}, {-82, 34}, {-82, 0}, {-74, 0}}));
  connect(T4.inPort, S2.outPort[2]) annotation(
    Line(points = {{24, -40}, {24, -40}, {24, -48}, {8, -48}, {8, -68}, {0, -68}, {0, -
68}, {0, -68}, {0, -68}}));
  connect(S2.outPort[1], T5.inPort) annotation(
    Line(points = {{0, -68}, {16, -68}, {16, -68}, {16, -68}}));
  connect(S2.inPort[1], T3.outPort) annotation(
    Line(points = {{-22, -68}, {-26, -68}, {-26, -46}, {-12, -46}, {-12, -38}, {-12, -
38}}));
  connect(breakWorkCycle.inPort[1], T5.outPort) annotation(
    Line(points = {{38, -68}, {22, -68}, {22, -68}, {22, -68}}));
  connect(T6.inPort, breakWorkCycle.outPort[1]) annotation(
    Line(points = {{62, -68}, {58, -68}, {58, -68}, {58, -68}}));
  connect(T1.inPort, S1.outPort[1]) annotation(
    Line(points = {{-44, 0}, {-54, 0}, {-54, 0}, {-54, 0}}));
  annotation(
    Diagram(graphics = {Rectangle(origin = {0, 1}, extent = {{-100, 99}, {100, -
101}})}),
    uses(Modelica(version = "3.2.3")));
end ControllerForTypicalScheme1;


model workCycleOfControllerForTypicalScheme1
  "Нормальний робочий цикл системи (кнопка start нажата)"
  extends Modelica.StateGraph.PartialCompositeStep;
```

```
  import SI = Modelica.SIunits;
  parameter SI.Height maxTank1_lvl "Введіть значення максимального рівня наповнення для Tank1";
  parameter SI.Height maxTank2_lvl "Введіть значення максимального рівня наповнення для Tank2";
  parameter SI.Height minTank1_lvl "Введіть значення рівня осаду для Tank1";
  parameter SI.Time fallingTime "Введіть час відстоювання рідини для Tank1 (секунд)";


  Modelica.StateGraph.Step openValve1 annotation(
    Placement(visible = true, transformation(origin = {-22, 70}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  Modelica.StateGraph.Step falling annotation(
    Placement(visible = true, transformation(origin = {-22, 38}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  Modelica.StateGraph.Transition T2( enableTimer = true, waitTime = fallingTime) annotation(
    Placement(visible = true, transformation(origin = {14, 38}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  Modelica.StateGraph.Step openValve2 annotation(
    Placement(visible = true, transformation(origin = {-22, 2}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  Modelica.StateGraph.Transition T3(condition = Tank1_lvl < minTank1_lvl or Tank2_lvl > maxTank2_lvl, enableTimer = false) annotation(
    Placement(visible = true, transformation(origin = {14, 2}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  Modelica.StateGraph.Step openValve3 annotation(
```

```
    Placement(visible = true, transformation(origin = {-22, -36}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  Modelica.Blocks.Interfaces.RealInput Tank1_lvl annotation(
    Placement(visible = true, transformation(origin = {-170, -80}, extent = {{-20, -
20}, {20, 20}}, rotation = 0), iconTransformation(origin = {-170, -80}, extent = {{-
20, -20}, {20, 20}}, rotation = 0)));
  Modelica.Blocks.Interfaces.RealInput Tank2_lvl annotation(
    Placement(visible = true, transformation(origin = {-170, -120}, extent = {{-20, -
20}, {20, 20}}, rotation = 0), iconTransformation(origin = {-170, -120}, extent = {{-
20, -20}, {20, 20}}, rotation = 0)));
  Modelica.StateGraph.Transition T4(condition = Tank2_lvl < 0.01, enableTimer =
false) annotation(
    Placement(visible = true, transformation(origin = {14, -36}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  Modelica.StateGraph.Step endOfStep annotation(
    Placement(visible = true, transformation(origin = {86, 0}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  Modelica.StateGraph.Transition T1(condition = Tank1_lvl > maxTank1_lvl)
annotation(
    Placement(visible = true, transformation(origin = {14, 70}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  Modelica.StateGraph.Step openValve4 annotation(
    Placement(visible = true, transformation(origin = {-22, -70}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  Modelica.StateGraph.Transition T5(condition = Tank1_lvl < 0.01, enableTimer =
false) annotation(
    Placement(visible = true, transformation(origin = {14, -70}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
equation
```

```
  connect(falling.inPort[1], T1.outPort) annotation(
    Line(points = {{-32, 38}, {-40, 38}, {-40, 54}, {40, 54}, {40, 70}, {16, 70}, {16,
70}}));
  connect(openValve2.inPort[1], T2.outPort) annotation(
    Line(points = {{-32, 2}, {-40, 2}, {-40, 20}, {40, 20}, {40, 38}, {16, 38}, {16,
38}}));
  connect(openValve3.inPort[1], T3.outPort) annotation(
    Line(points = {{-32, -36}, {-40, -36}, {-40, -20}, {40, -20}, {40, 2}, {16, 2}, {16,
2}}));
  connect(openValve4.inPort[1], T4.outPort) annotation(
    Line(points = {{-32, -70}, {-40, -70}, {-40, -52}, {40, -52}, {40, -36}, {16, -36},
{16, -36}}));
  connect(T5.outPort, endOfStep.inPort[1]) annotation(
    Line(points = {{16, -70}, {60, -70}, {60, 0}, {76, 0}, {76, 0}}));
  connect(inPort, openValve1.inPort[1]) annotation(
    Line(points = {{-160, 0}, {-80, 0}, {-80, 70}, {-32, 70}, {-32, 70}}));
  connect(T5.inPort, openValve4.outPort[1]) annotation(
    Line(points = {{10, -70}, {-12, -70}, {-12, -70}, {-12, -70}}));
  connect(T4.inPort, openValve3.outPort[1]) annotation(
    Line(points = {{10, -36}, {-11.5, -36}}));
  connect(T3.inPort, openValve2.outPort[1]) annotation(
    Line(points = {{10, 2}, {-11.5, 2}}));
  connect(T2.inPort, falling.outPort[1]) annotation(
    Line(points = {{10, 38}, {-11.5, 38}}));
  connect(T1.inPort, openValve1.outPort[1]) annotation(
    Line(points = {{10, 70}, {-11.5, 70}}));
  connect(endOfStep.outPort[1], outPort) annotation(
    Line(points = {{96.5, 0}, {156, 0}}));
  annotation(
```

```
    Diagram(coordinateSystem(initialScale = 0.1)),
    uses(Modelica(version = "3.2.3")));
end workCycleOfControllerForTypicalScheme1;
```